Formally, a **search problem** has 5 components:

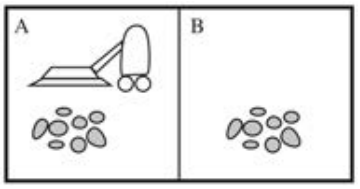
- s_0: Initial state
- Actions(s): Returns the set of actions **applicable** in state s. (E.g. the action of opening a door might only be applicable if the door is unlocked.)
- Results(s, a): Returns the state s' reached from s by executing action a.
- Goal-Test(s): Returns `true` if s is goal state, otherwise `false`.
- Step-Cost(s, a): The cost of executing action a in s. Most often we will assume Step-Cost(s, a) $= 1$ for all s and a.

A state g is called a **goal state** if Goal-Test(g) $=$ `true`.

A **solution** to a search problem is a **sequence of actions** (a **path**) from s_0 to a goal state. It is **optimal** if it has minimum sum of step costs.

A Toy Problem: Vacuum World

Vacuum World consists of two locations, each of which may or may not contain dirt and the vacuum is in one of the locations.



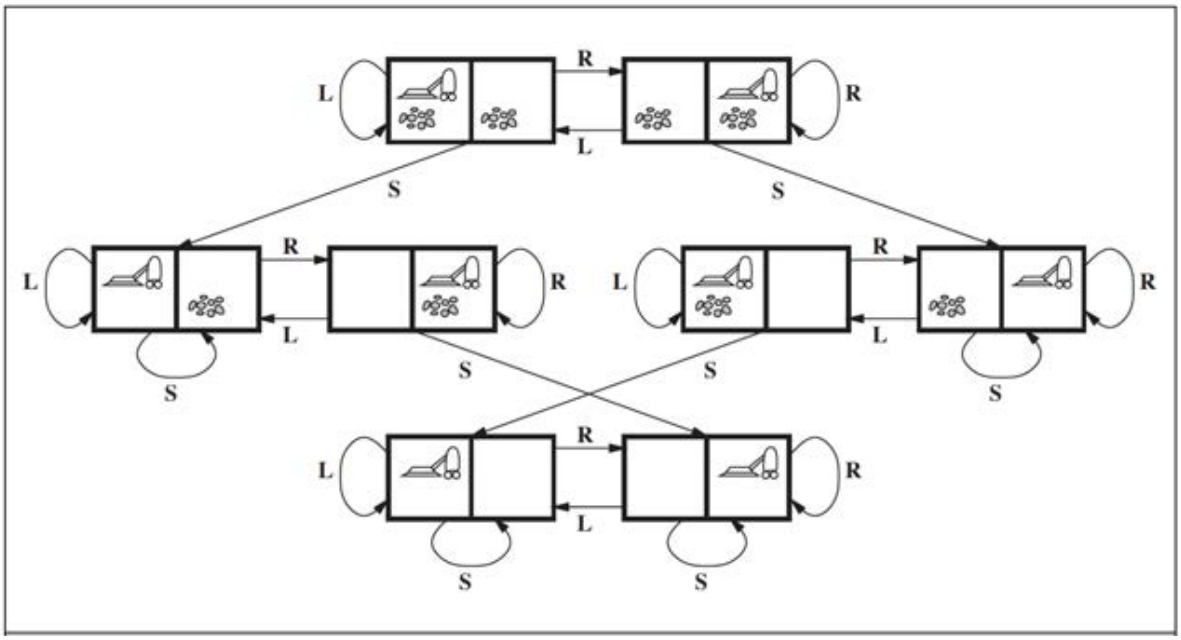
States space consists of each possible configuration (2×2^2 possible states).

- s_0: Initial state
- Actions(s): for each state three possible actions: L, R, S.
- Results(s, a): actions have their expected results.
- Goal-Test(s): *are all squares clean?*
- Step-Cost(s, a): each step costs 1.

A Toy Problem: Vacuum World

Transition Model

The transitions in the Vacuum World can be represented as a graph.



States include a location and the current time, etc.

- s_0: specified by the user's query.
- Actions(s): take any flight from the current location, in any class, etc.
- Results(s, a): the state resulting from taking a flight.
  flight destination as current location and flight arrival time as current time.
- Goal-Test(s): *Are we at the final destination specified by the user?*
- Step-Cost(s, a): monetary cost, waiting time, flight time, customs and immigration procedures, seat quality, time of day, type of airplane, frequent-flyer mileage awards, etc.

Expanded nodes & frontier

In tree and graph search, we always generate **all** children of a chosen state s:
We compute Result(s, a) for *all* applicable actions a.
We call this process **expanding** s.

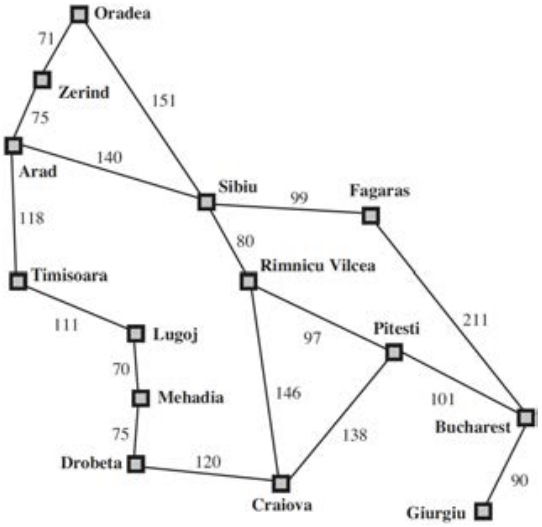
In tree search states are called **nodes**: In tree search we don't keep track of repeated states, and hence distinct tree nodes might represent the same state.

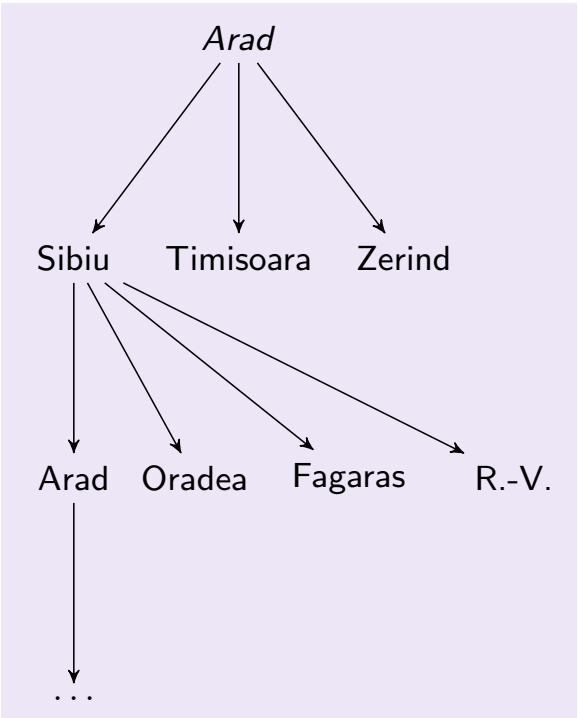
The search has two types of nodes:

- **Expanded nodes**: Nodes for which all children have been generated.
- **Frontier**: Nodes that we generated, but not expanded.

function Tree-Search (problem) returns a solution, or failure
  frontier := $\{s_0\}$ (initial state) // we initialise the frontier
  loop do
    if *frontier* = $\emptyset$ **then return** failure
    choose a node *n* from *frontier*
    remove *n* from *frontier*
    if *n* is a goal state **then return** solution
    for each child *m* of *n* // we expand *n*
      add child m to frontier

From Tree-search to Graph-search





Failure to detect repeated states can lead to infinite loops in Tree-Search.
Solution: Keep track of already generated states.
Adding this check to Tree-Search gives us Graph-Search.

The red lines are those added to the Tree-Search algorithm.

function Graph-Search (problem) returns a solution, or failure
  expandedNodes := ∅
  frontier := $\{s_0\}$ (initial state) // we initialise the frontier
  loop do
    if *frontier* = ∅ **then return** failure
    choose a node *n* from *frontier*
    remove *n* from *frontier*
    add *n* to *expandedNodes*
    if *n* is a goal state **then return** solution
    for each child *m* of *n* // we expand *n*
      if *m* ∉ *frontier* and *m* ∉ *expandedNodes* then
        add child m to frontier

Solving search problems

Solutions to search problems split into:

- **Tree search**
- **Graph search**

The only difference is: graph search keeps track of repeated states.

Along a different dimension they split into:

- **Uninformed search**: No sense of closeness to the goal.
- **Informed search**: Heuristics used to give a sense of closeness to the goal (e.g. straight-line distance to goal in case of route finding on a map).

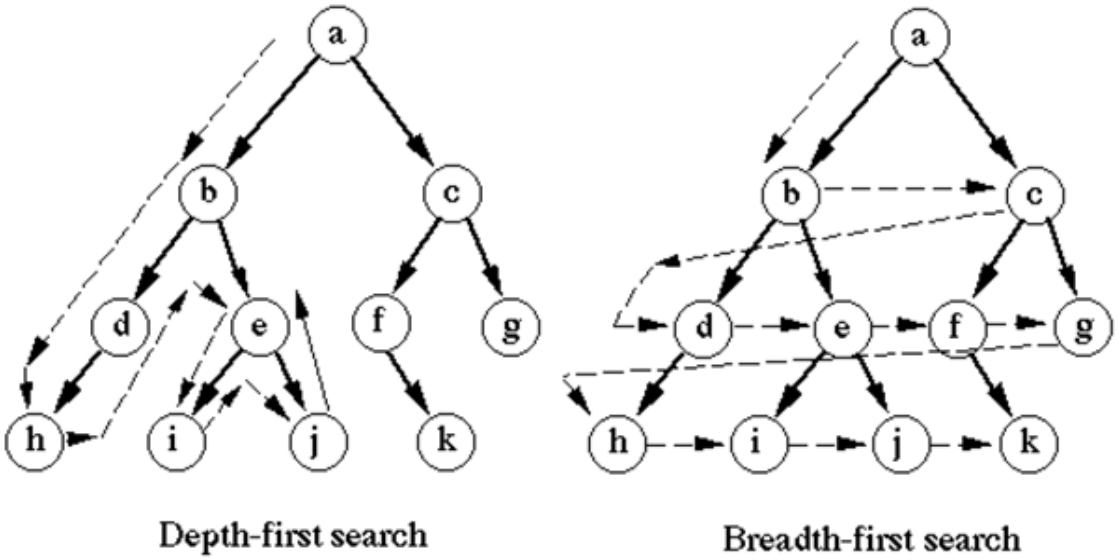
For now, we only consider uninformed search.

Along yet another dimension:

- **Breadth-based search**: Exploring shallow nodes first (close to s_0).
- **Depth-based search**: Exploring as deeply as possible, sacrificing breadth (might miss the goal).

DFS: Depth-first search. **BFS**: Breadth-first search.

Order of node expansion:



Search strategies of Tree-Search and Graph-Search

Different search strategies can be achieved by simply changing how choose node from frontier and add child to frontier work.

Breadth-first search (BFS):

- Frontier is **queue** (FIFO).
- choose node from frontier: dequeue node from frontier.
- add child to frontier: enqueue node to frontier.

Depth-first search (DFS):

- Frontier is **stack** (LIFO).
- choose node from frontier: pop node from frontier.
- add child to frontier: push node to frontier.

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: [a] complete if b is finite; [b] complete if step costs $\geq \epsilon$ for positive ϵ; [c] optimal if step costs are all identical; [d] if both directions use breadth-first search.

Problem solving by searching

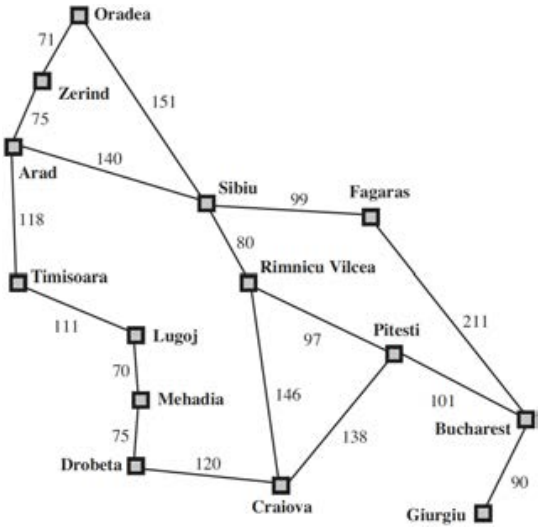
Problem solving is an essential part of intelligence.

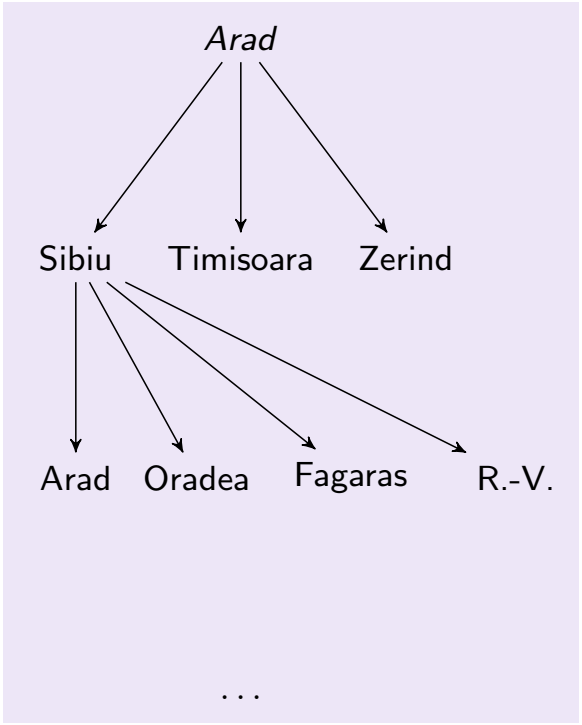
Problem solving can be represented as a search strategy.

Uniformed searching is **blindly** following transitions from states to states.

Informed search uses extra information in making choices of transitions.

This information is given in the form of **heuristic function**.





The Tree-Search algorithm

function Tree-Search (problem) returns a solution, or failure
  frontier := $\{s_0\}$ (initial state) // we initialise the frontier
  loop do
    if *frontier* = $\emptyset$ **then return** failure
    choose a node *n* from *frontier*
    remove *n* from *frontier*
    if *n* is a goal state **then return** solution
    for each child *m* of *n* // we expand *n*
      add child m to frontier

Different search strategies can be achieved by simply changing how choose node from frontier and add child to frontier work.

Breadth-first search (BFS):

- Frontier is **queue** (FIFO).
- choose node from frontier: dequeue node from frontier.
- add child to frontier: enqueue node to frontier.

Uniform-cost search

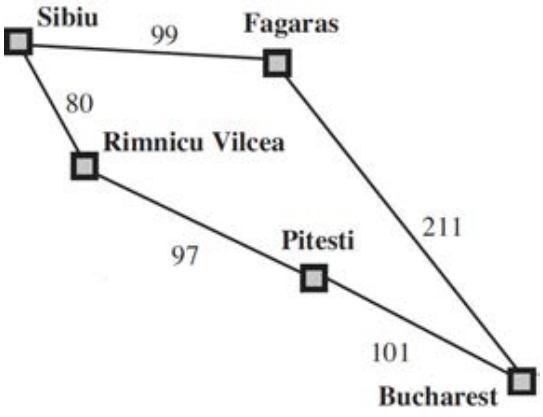
When all step-costs are equal BFS is optimal.
UCS is an extension of BFS which is optimal for any step-cost function.

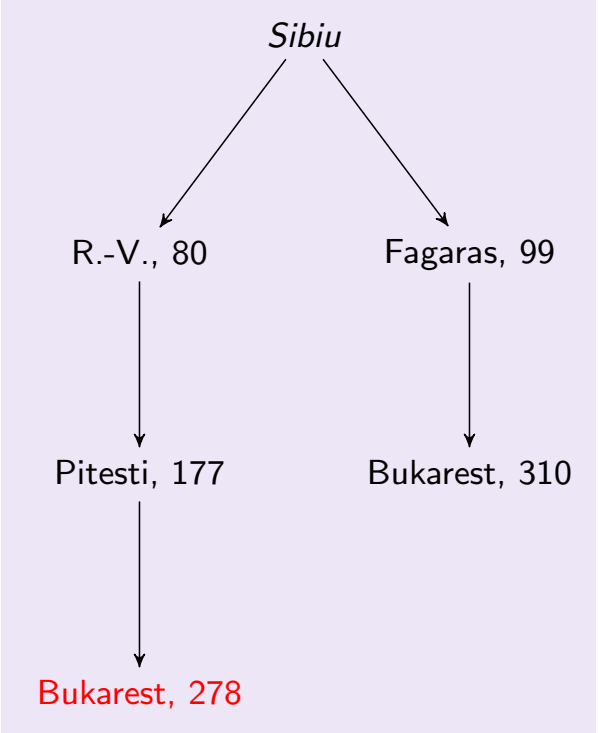
Instead of expanding the shallowest node
UCS expands the node n with the lowest path-cost g(n).

Frontier is a priority queue ordered by g.

Differences with BFS:

- Goal-Test is applied when a node is selected for expansion
  (rather than when it is first generated)
  (the first goal node that is generated may be on a suboptimal path)
- a test is added for a better path to a node currently on the frontier





Best-first search

Best-first search is an instance of the general TREE- or GRAPH-SEARCH where node selection for expansion is based on an **evaluation function**, f.

The function f is a cost estimate: node n with lowest $f(n)$ is expanded first.

Best-first graph search the same as UCS but with f for priority queue.

The choice of f determines the search strategy.

Most best-first algorithms include as a part of f a heuristic function,

$h(n)$ = estimated cost of the cheapest path from node n to a goal state.

Heuristic functions are the most common form in which additional knowledge of the problem is imparted to the search algorithm.

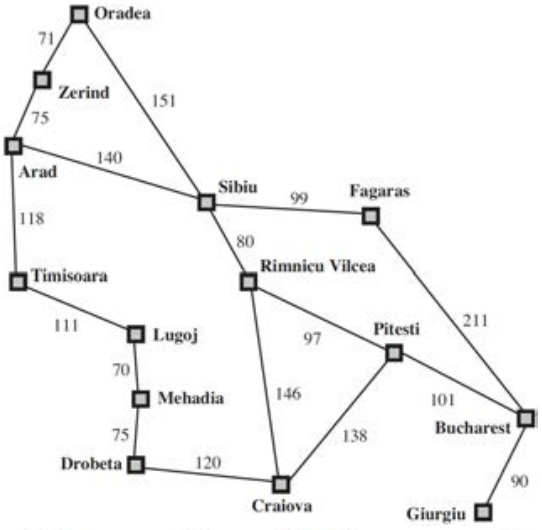
Heuristics are arbitrary, nonnegative, problem-specific functions, with one constraint: if n is a goal node, then $h(n) = 0$.

Greedy best-first search

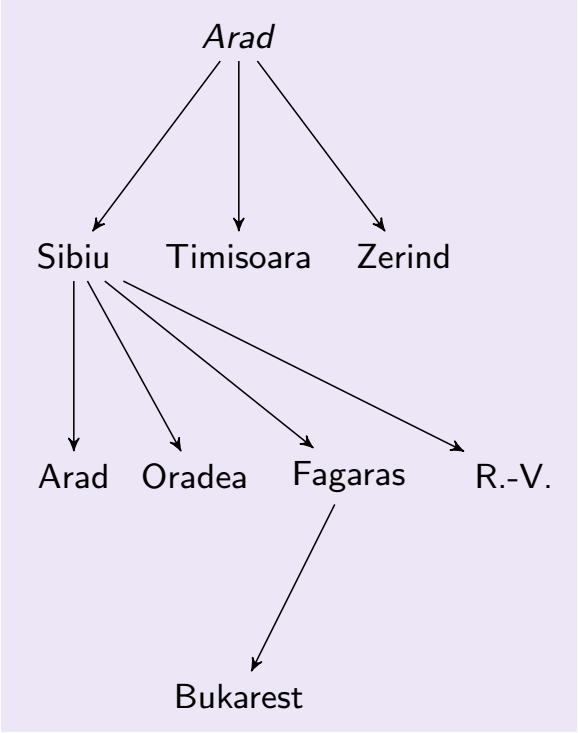
Greedy best-first search tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly.

Greedy best-first search evaluates nodes by using just the heuristic function:

$$f(n) = h(n).$$



Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



Properties of Greedy Best-First Search

- It's greedy.
- It's incomplete.
- The complexity (for tree version) is as DFS.

A⋆ evaluates a node n by combining:

- $g(n)$, the cost to reach the node, and
- $h(n)$, the estimated cost to get from the node to the goal:
- $f(n) = g(n) + h(n)$.

Provided that the heuristic function $h(n)$ satisfies certain conditions,

A⋆ search is both complete and optimal.

The A⋆ algorithm is UCS except that it uses $g + h$ instead of g.

Designing heuristics

Designing good heuristic function $h(n)$ is in many cases a great art form.

Q: What properties should a good heuristic function have?

A: Estimate cost to goal as precisely as possible; be as cheap as possible to compute (cheaper than computing the actual cost).

Recall: STEP-COST(s, a) is the **cost** of executing a in s.

Optimal cost of a node n:
The minimal cost to achieve a goal from n, denoted $h^*(n)$.

Admissible heuristics: cost of reaching the goal is never overestimated:
the heuristics is **always optimistic**. Formally: $h(n) \leq h^*(n)$ for all nodes n.

Optimal search algorithm: The algorithm always returns an **optimal solution**:
a solution of minimal cost.

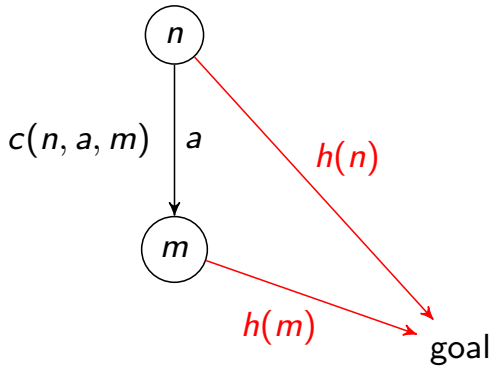
Theorem.
$A^\star$ TREE-SEARCH is optimal when h is admissible.

CONSISTENCY

Let $c(n, a, m)$ denote the STEP-COST of executing action a to go from n to m.

Consistent heuristics: If m is reached by executing a in n then

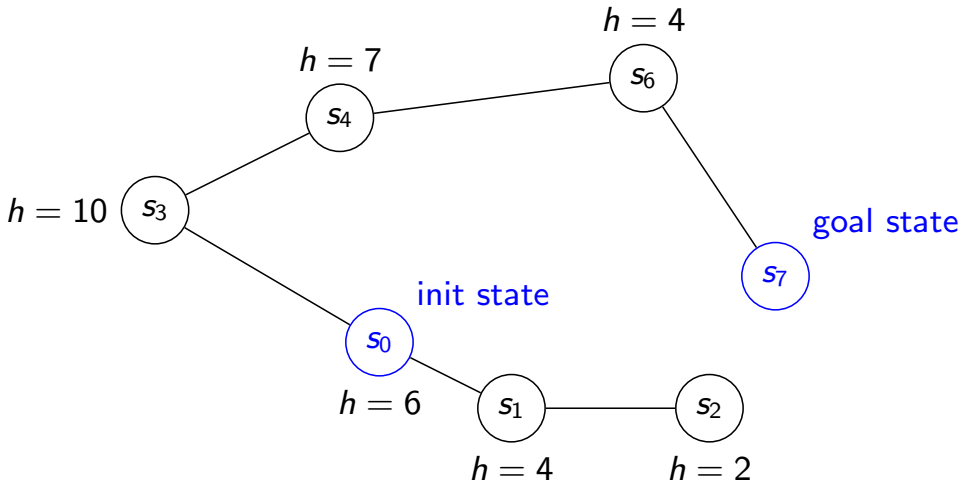
$$h(n) \leq c(n, a, m) + h(m)$$



Theorem. Consistency $\Rightarrow$ admissibility.

Theorem. $A^\star$ GRAPH-SEARCH is optimal when h is consistent.
(See R&N, Chapter 3, for the discussion and proof outlines.)

Consider the state space below. All step costs are 1.



Q1 How does greedy best-first tree search behave on the problem?

Q2 How about greedy best-first graph-search?

Q3 And A* tree-search?

Dominating heuristics

A heuristics h_2 is said to **dominate** a heuristics h_1 if

$$h_2(s) \geq h_1(s) \text{ for all } s.$$

If h_2 dominates h_1 and both are admissible, $A^\star$ will never expand more nodes using h_2 than using h_1. So h_2 is better in that sense.

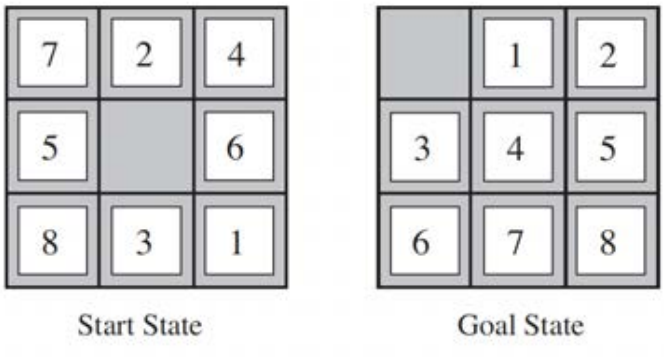
If $h_1, \ldots, h_n$ are admissible heuristics, then so is $h(s) = \max\{h_1(s), \ldots, h_n(s)\}$.

It is always better to use h as heuristics than either of the h_i,
unless the penalty in computation time of h is too high.

Often heuristics are generated via **relaxed problems**.

Relaxed problem:
A simplified version of a problem with fewer restrictions.
A solution to the original problem is also a solution to the relaxed problem.

Example



Relaxing the 8-puzzle:

1. A tile can move to any adjacent square.
2. A tile can move to any square.

Generating heuristics via relaxed problems

Take a problem P and let $h^*_P(n)$ be the optimal cost to get to goal from n in P.

Given any problem P and relaxation P', a heuristics h for P can be defined by:

$$h(n) = h^*_{P'}(n).$$

In words: The **estimated cost** of a solution to the **real problem** is taken to be the **actual cost** of a solution to the **relaxed problem**.

Q1 Why is h defined above admissible?

Q2 Why is h defined above consistent?

Q3 Which heuristics do we get from the sliding puzzle relaxations of the previous slide (1. Move to any adjacent square; 2. Move to any square)?

Q4 Does one of the heuristics from the previous question dominate the other?

Q5 Given P', how do we calculate $h^*_{P'}$?

So far we have only considered search problems in environments that are:

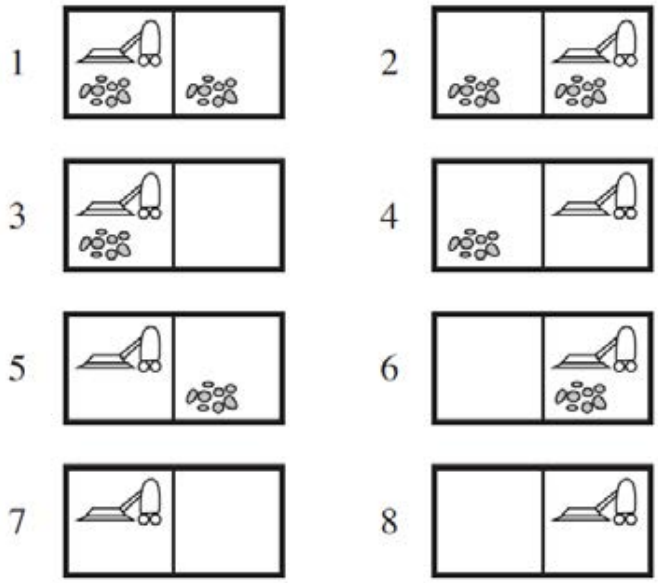
- **Single-agent**. There is a single agent acting, the one we control.
- **Static**. When the agent is not acting, the world doesn't change.
- **Deterministic**. Every action has a unique outcome.
- **Fully observable**. The full state description is accessible to the agent.

Problem solving in the real world rarely satisfies these assumptions.

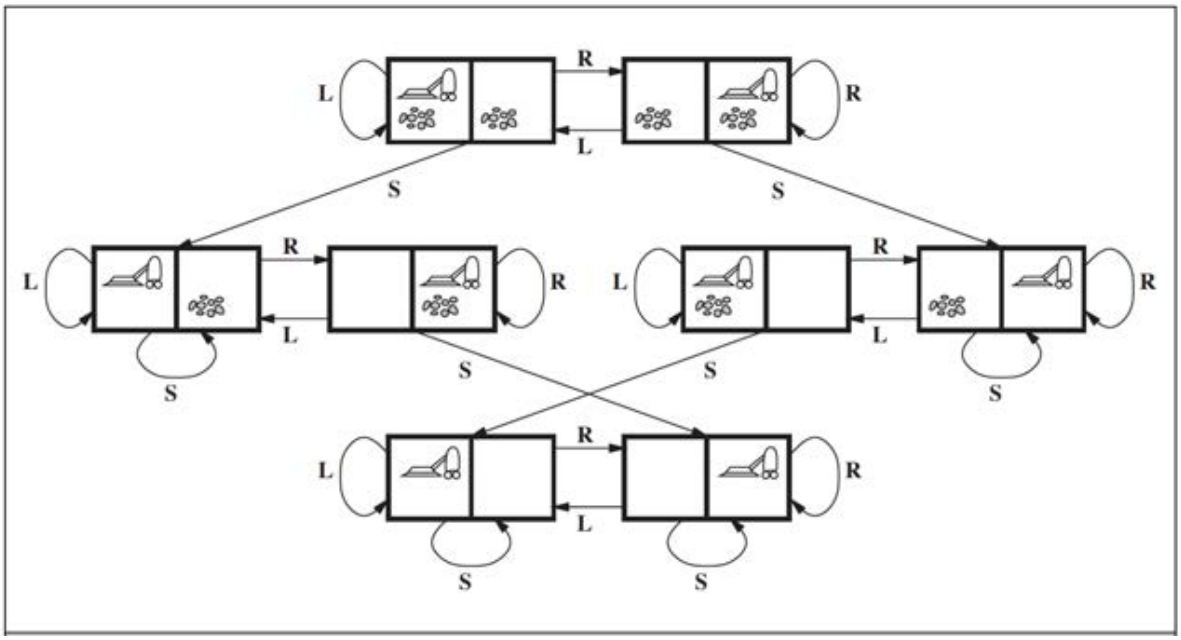
Today, we will drop the assumption of **determinism** and **full observability**.

Recap: Vacuum World

Possible states



The transitions in the Vacuum World can be represented as a graph.



Outline

Non-determinism

Partial Observability

Let us consider an erratic vacuum:

- When applied to a dirty square it cleans the square
  and sometimes cleans up dirt in an adjacent square, too.
- When applied to a clean square it sometimes deposits dirt in that square.

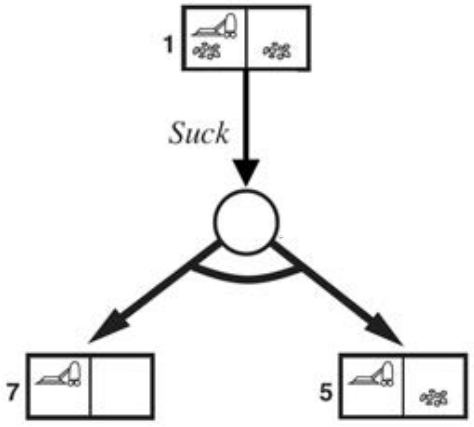
Erratic Vacuum World
Problem Description

- s_0: Initial state
- Actions(s): for each state three possible actions: L, R, S.
- Results(s, a): applying a in s leads to **a set of states** S'.
- Goal-Test(s): *are all squares clean?*
- Step-Cost(s, a): each step costs 1.

Q: What is a solution to the erratic vacuum problem starting in the state 1?

Search problems with nondeterminism: Results(s, a) returns a *set* of states.

Example. Results$(1, \mathit{Suck}) = \{5, 7\}$.



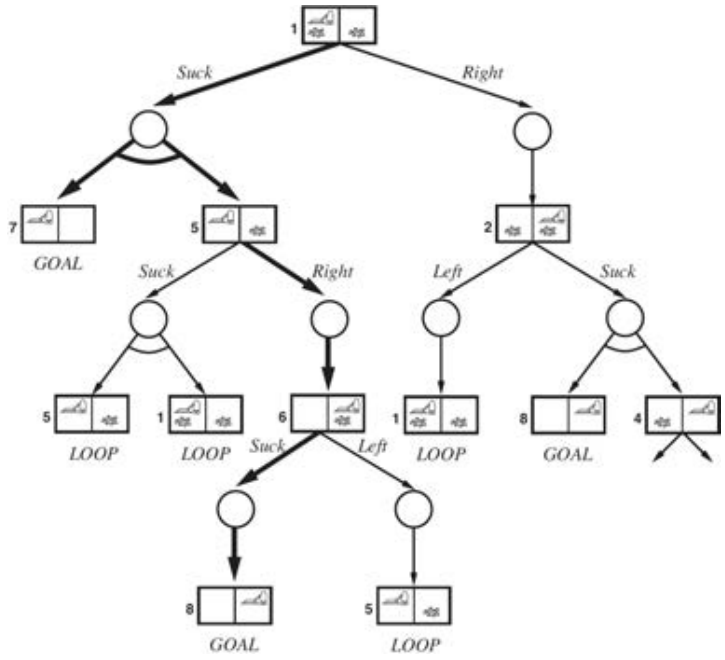
Non-deterministic action is represented with an **AND node**: outgoing edges representing all outcomes **linked by an arc**.

OR nodes are as usual, with deterministic Actions.

And-or search trees

AND-OR tree is a tree with levels of AND and OR nodes.

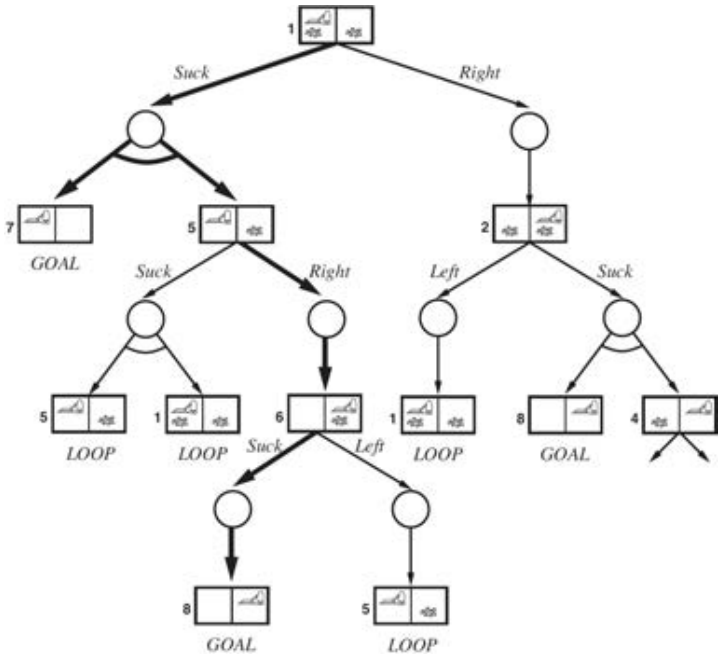
- Root is OR node (agent choice).
- OR nodes have outgoing edges for all applicable actions.
- AND nodes have outgoing edges for all outcomes of action.



Solution to a Non-deterministic Search Problem

A **solution** to a nondeterministic search problem is a subtree T' of T s.t.:

1. The root note of T is in T'.
2. Every leaf of T' is a goal state.
3. Every OR node of T' has exactly one outgoing edge (agent choice).
4. Every AND node of T' has the same outgoing edges as in T (nature choice).



Solutions as conditional plans and policies

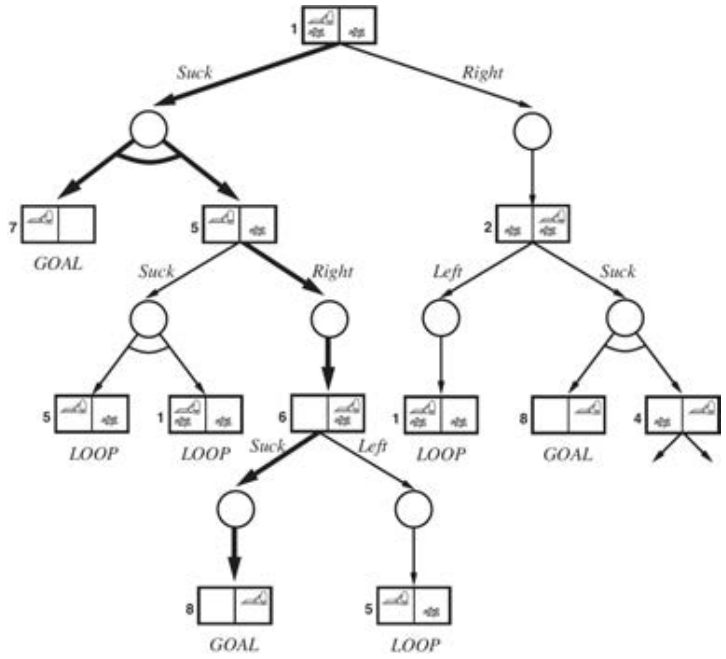
Deterministic search problems: Solution is a path in state space
it can be turned into a sequence of Actions (sequential plan).

Non-deterministic search problems: Solution is a subtree of state space,
it can be turned into a conditional plan (contingency plan).

Language of **conditional plans**:

$$\pi ::= \varepsilon \mid a \mid \text{ if } s \text{ then } \pi_1 \text{ else } \pi_2 \mid \pi_1; \pi_2$$

where ε is the empty plan, $a \in$ Actions and s is a state.
The construct $\pi_1; \pi_2$ denotes sequential composition: first execute π_1, then π_2.



T' as a **conditional plan**: $\pi = \mathit{Suck}$; if s_5 then (Right; Suck).

T' as a **policy** (a mapping from states to Actions):
Policy Π: $\Pi(s_1) = \mathit{Suck}$, $\Pi(s_5) = \mathit{Right}$, $\Pi(s_6) = \mathit{Suck}$.

AND-OR graph search

function And-Or-Graph-Search(*problem*) returns a conditional plan, or failure
  Or-Search(*init state of problem*, []) // problem is implicit parameter

function Or-Search(*state*, *path*)
  if *state* is a goal **then return** ε // if in goal state, empty plan suffices
  if *state* is on *path* **then return** *failure* // fail if looping
  for each *action* applicable in *state* **do** // recursively search for plan
    plan $\leftarrow$ And-Search(Results(*state*, *action*), [*state* | *path*])
    if *plan* $\neq$ *failure* **then return** *action*; *plan* // append plan to action
  return *failure* // if all recursive searches for a plan fails

function And-Search(*states*, *path*)
  for each s_i **in** *states* **do** // recursively find plans for each outcome state
    $\mathit{plan}_i \leftarrow$ Or-Search(s_i, *path*)
    if plan_i = *failure* **then return** *failure*
  return if s_1 then plan_1 else if s_2 then plan_2 else $\cdots$ if s_{n-1} then plan_{n-1} else
    plan_n // if $n = 1$ then the returned plan is just plan_1

function AND-OR-GRAPH-SEARCH(*problem*) returns a conditional plan, or failure
  OR-SEARCH(*init state of problem*, []) // problem is implicit parameter

function OR-SEARCH(*state*, *path*)
  if *state* is a goal **then return** ε // if in goal state, empty plan suffices
  if *state* is on *path* **then return** *failure* // fail if looping
  for each *action* applicable in *state* **do** // recursively search for plan
    $plan \leftarrow$ AND-SEARCH(RESULTS(*state*, *action*), [*state* | *path*])
    if $plan \neq failure$ **then return** *action*; *plan* // append plan to action
  return *failure* // if all recursive searches for a plan fails

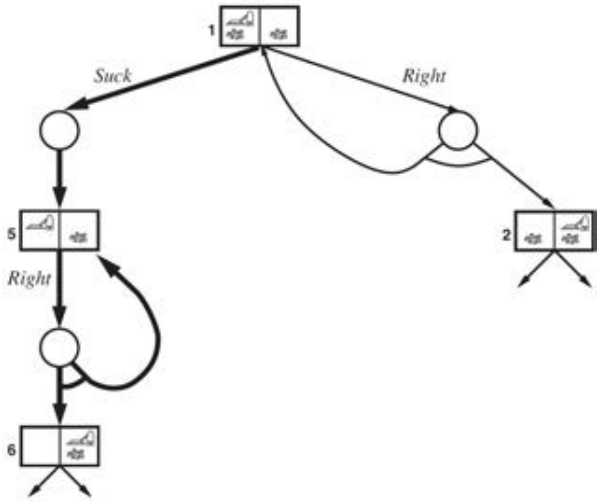
function AND-SEARCH(*states*, *path*)
  for each s_i **in** *states* **do** // recursively find plans for each outcome state
    $plan_i \leftarrow$ OR-SEARCH(s_i, *path*)
    if $plan_i = failure$ **then return** *failure*
  return if s_1 then $plan_1$ else if s_2 then $plan_2$ else $\cdots$ if s_{n-1} then $plan_{n-1}$ else
      $plan_n$ // if $n = 1$ then the returned plan is just $plan_1$

Q: Which algorithm do we get when all ACTIONS are deterministic?

THE SLIPPERY VACUUM WORLD

Let us consider a slipping vacuum:

- everything normal, except that
- movement action sometimes fails.



Acyclic solutions are not always possible.

Cyclic solutions: Subtrees T' where every leaf is either a goal or a loop node.

Cyclic solutions guarantee reaching the goal, but only when assuming **fairness**: Every possible outcome of an action will eventually happen.

We need a language of conditional plans with loops.

$$\pi ::= \varepsilon \mid a \mid \text{ if } s \text{ then } \pi_1 \text{ else } \pi_2 \mid \pi_1; \pi_2 \mid \text{ while } cond \text{ do } \pi_1$$

where *cond* is some logical condition.

The slipping vacuum: *Suck* ; *Right*; while *not in* 6 do *Right*; *Suck*

Limiting Observability

We are now back to deterministic domains.

We considered searching under **full observability**:
the full state description is observable by the agent.

Under **partial observability** the full state might not be observable:
e.g. vacuum cleaning robot not knowing which squares are clean.

Null observability: is the case when *nothing* can be observed about the states.
Search problems under null observability are called
conformant problems or **sensorless problems**.

Example. Consider the vacuum where it is not known which squares are clean, and the robot doesn't have any sensors.

Q: Can the problem still be solved, and if so, what is the solution?

Belief state: A set of (physical) states.
Contains the states considered possible by an agent in a given state.

We generally use s for (physical) states and b for belief states.

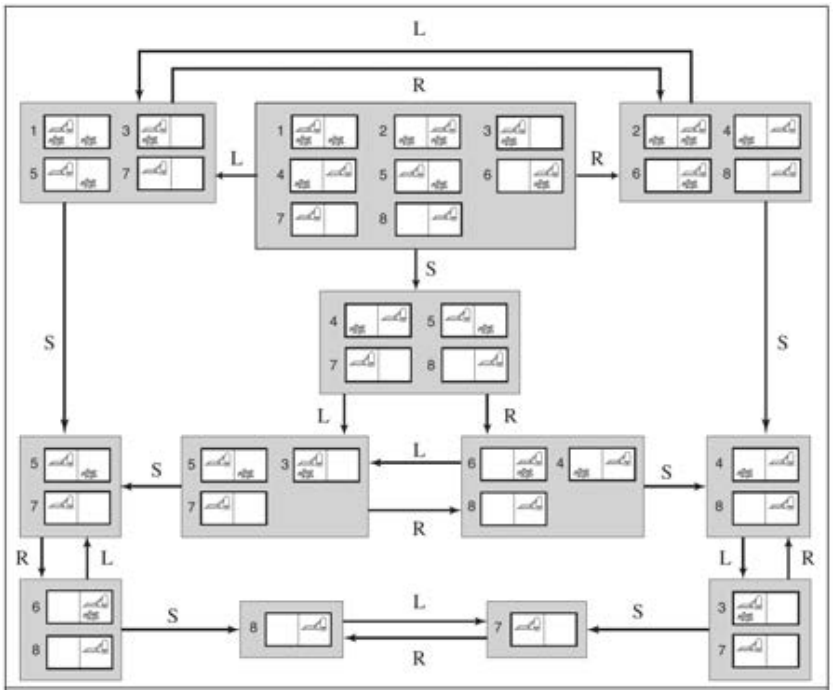
If a problem has n states, there can be up to 2^n belief states:
an exponential blow-up in the worst case.

Conformant problems can be solved by
any of the standard graph and tree search algorithms (e.g. $A^\star$),
just using belief states instead of physical states.

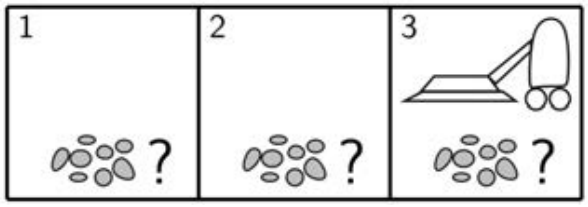
Belief states

Formally, given a fully observable problem
$(s_0, \textsc{Actions}, \textsc{Results}, \textsc{Goal-Test})$,
we can define a corresponding conformant problem
$(b_0, \textsc{Actions}', \textsc{Results}', \textsc{Goal-Test}')$
with initial belief state b_0 by:

$$\textsc{Actions}'(b) = \bigcup_{s \in b} \textsc{Actions}(s)$$
$$\textsc{Results}'(b, a) = \bigcup_{s \in b} \textsc{Results}(s, a)$$
$$\textsc{Goal-Test}(b) = \bigwedge_{s \in b} \textsc{Goal-Test}(s)$$



Searching with Observations (Sensing)



Assume the robot doesn't initially know which squares are dirty,
but it has a *Sense* action to check whether the current square is clean or dirty.

Suppose the number of *Suck* actions have to be minimised.

Q1: What would then be a solution to the cleaning problem?

Q2: Can we use the Graph-Search to solve problems with partial observability and sensing?

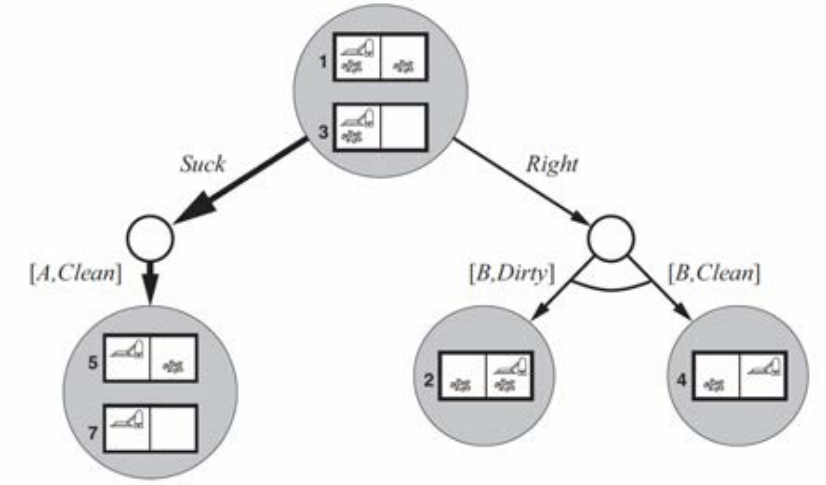
Q3: And what about And-Or-Graph-Search?

A general treatment of observations/sensing under partial observability:

including a new function in the problem description: $\textsc{Percept}(s)$
which codes the observation made by the agent in state s.

Example: $\textsc{Percept}(s) = \textit{dirty}/\textit{clean}$.

Full observability corresponds to $\textsc{Percept}(s) = \{s\}$.
Null observability corresponds to $\textsc{Percept}(s) = \emptyset$.



The new RESULTS function

We define the following new functions:

1. Possible-Percepts takes a belief state b and returns all the observations that are possible to receive in that belief state:

$$\textsc{Possible-Percepts}(b) = \{\textsc{Percept}(s) \mid s \in b\}.$$

2. Update takes a belief state b and an observation o and filters away the states that are not consistent with the observation:

$$\textsc{Update}(b, o) = \{s \in b \mid o = \textsc{Percept}(s)\}.$$

   So $\textsc{Update}(b, o)$ is the updated belief an agent has after having received observation o in belief state b.

New Results function on belief states that takes observations into account:

$$\textsc{Results}'(b, a) = \begin{array}{l}\{\textsc{Update}(\bigcup_{s\in b} \textsc{Results}(s, a), o) \mid \\ o \in \textsc{Possible-Percepts}(\bigcup_{s\in b} \textsc{Results}(s, a))\}\end{array}$$

Note that $\textsc{Results}(b, a)$ is a set of belief states, that is, a set of sets of states.

Every day examples of interactions:

- Driving in traffic.
- Bargain-hunting auctioning.
- Governmental elections.

Historical examples:

- Anthony and Cleopatra.
- Hitler and Stalin.
- Kruschev and Kennedy.

Game theory 'works' when agents behave **rationally**.

Game Theory

A branch of applied mathematics used in: social sciences, economics, biology, linguistics, engineering, political science, international relations, computer science, philosophy, etc.

Game theory attempts to **mathematically** capture
behavior in **strategic** situations,
in which an individual's success in making choices
depends on the choices of others.

Initially developed for competitive scenarios, later extended to cooperative ones.

> *Game theory is a sort of umbrella or 'unified field' theory for the rational side of social science, where 'social' is interpreted broadly, to include human as well as non-human players (computers, animals, plants).*
>
> *(Aumann 1987)*

A **game** is a description of a situation of interaction of two or more agents.

It includes:

- the players;
- the actions that players can take;
- the preferences of the agents over the possible outcomes of the game.

A **strategic game** is a game in which players involved make one decision each, and make it independently.

Toy Games: Matching Pennies

- Two players: Alice and Bob.
- Both show a coin.
- Bob wins if they show different faces.
- Alice wins if they show the same.
- they each have two strategies: *heads* and *tails*.
- All possible strategy combinations give payoffs for each player.

Toy Games: Matching Pennies (conflict)

Alice	heads	tails
heads	+	−
tails	−	+

Bob	**heads**	**tails**
heads	−	+
tails	+	−

Together:

	heads	tails
heads	(+,−)	(−,+)
tails	(−,+)	(+,−)

Toy Games: Driving Game (cooperation)

Alice	left	right
left	+	−
right	−	+

Bob	**left**	**right**
left	+	−
right	−	+

Together:

	left	right
left	(+,+)	(−,−)
right	(−,−)	(+,+)

	heads	tails
heads	(1,-1)	(-1,1)
tails	(-1,1)	(1,-1)

	left	right
left	(1,1)	(-1,-1)
right	(-1,-1)	(1,1)

Table: Matching Pennies and Driving Game with payoffs -1 and 1

Any numbers as long as winning>losing.

Zero-sum Games

Recall Matching Pennies game:

	heads	tails
heads	(1,-1)	(-1,1)
tails	(-1,1)	(1,-1)

Payoffs in each cell sum up to 0.
This can be always done for games with pure conflict.

Games, esp. board games became, the playground for the development of AI.

The focus on **zero-sum**, **turn-based games**, e.g., Chess or Go.

The games of interest are often also **perfect-information** games.

Search Problems Formally

So far, search problems were defined by:

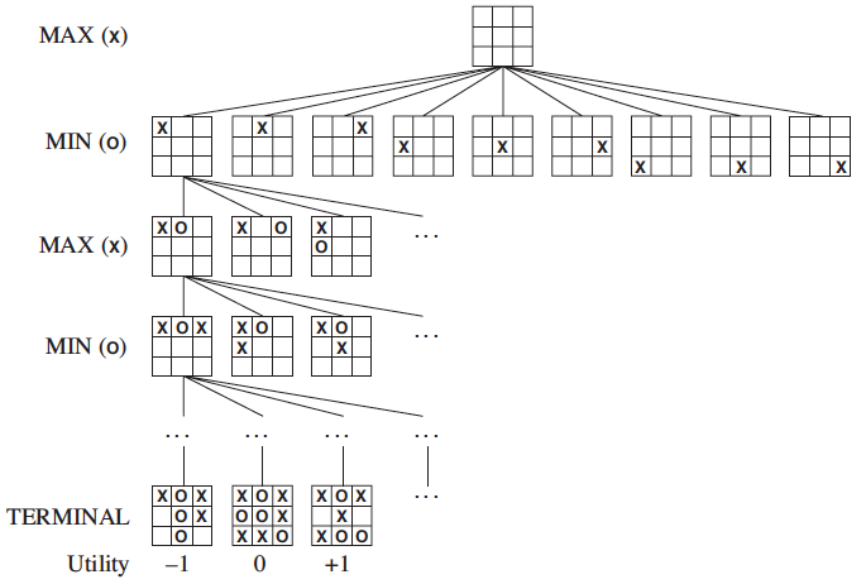
- **Initial state**.
- Actions(s): **possible actions**.
- Result(s, a): **transition model**.
- **Goal test**.

That induces a **state space** in which we search for a goal state.

Similarly, games can be described by:

- s_0: **initial state**.
- Player(s): who has the move in s.
- Actions(s): Legal moves in s.
- Result(s, a): **transition model**.
- Terminal-Test(s): **terminal test**. Is the game over?
- Utility(s, p): **utility function** (or **payoff function**).
  Numerical value for player p in terminal state s.
  Example: $+1$ for win and -1 for loose (zero-sum).

Tic-Tac-Toe: Game Tree Example



- There are two players: MAX and MIN.
- The game is zero-sum.
- Nodes assigned values representing expected utility for MAX.

$$\textsc{Minimax}(s) = \begin{cases} \textsc{Utility}(s, \textsc{max}) & \text{if } \textsc{Terminal-Test}(s) \\ \max_{a \in \textsc{Actions}(s)} \textsc{Minimax}(\textsc{Result}(s, a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_{a \in \textsc{Actions}(s)} \textsc{Minimax}(\textsc{Result}(s, a)) & \text{if } \textsc{Player}(s) = \textsc{min} \end{cases}$$

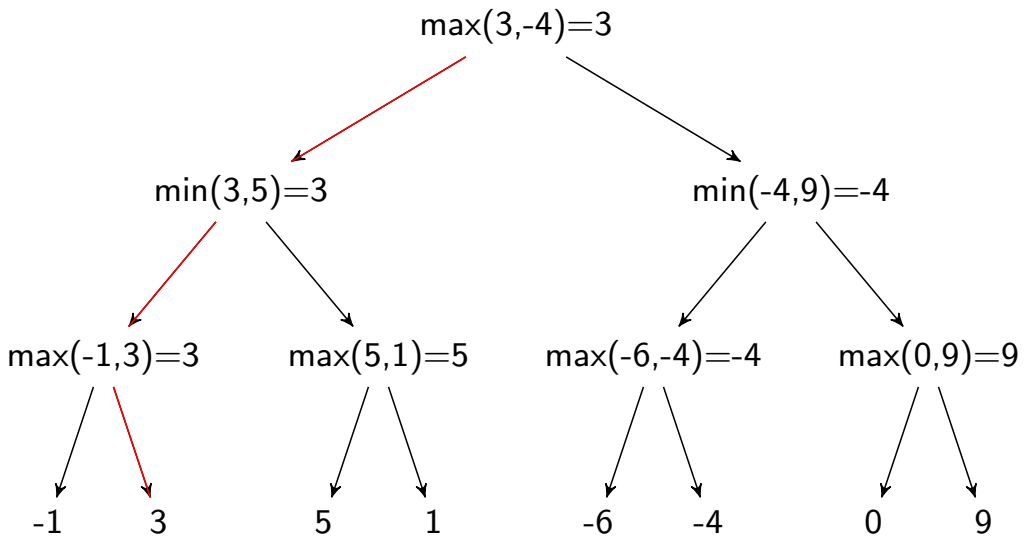
In a state s with $\textsc{Player}(s) = \textsc{max}$, the following move is optimal for MAX:

$$\operatorname{argmax}_{a \in \textsc{Actions}(s)} \textsc{Minimax}(s).$$

That is, MAX chooses the move that maximises the minimax value.

Is MINIMAX a breadth- or a depth-first search?

MINIMAX ON AN EXAMPLE



Alpha-beta pruning can make the game tree smaller while still guaranteeing optimal strategies.

The recursive Minimax algorithm **passes down** an α and β value to each node:

- α: **lower bound** on what MAX can achieve when playing through the choice points leading to the current node.
- β: **upper bound** on what MIN can achieve when playing through the choice points leading to the current node.

The root node has $\alpha = -\infty$ and $\beta = \infty$.

Alpha-beta Pruning

Additionally, the algorithm iteratively updates the value v of each node. Initially $v = -\infty$ for MAX nodes and $v = \infty$ for MIN nodes.

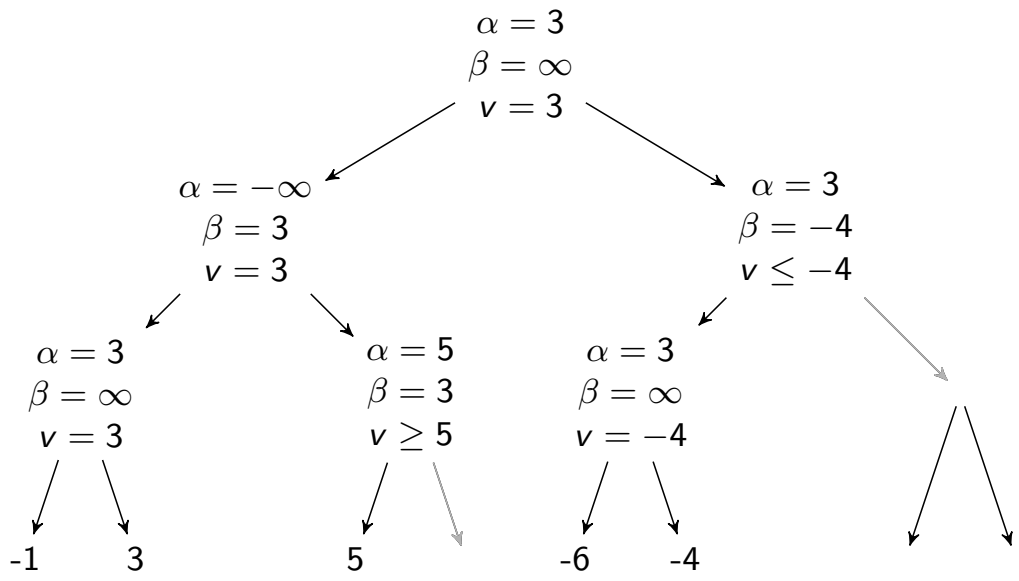
- **Alpha-cut**: If $v \leq \alpha$ in a MIN node, we can prune further search below that node: MAX has a better choice at a previous choice point.
- **Beta-cut**: If $v \geq \beta$ in a MAX node, we can prune further search below that node: MIN has a better choice at a previous choice point.

α-values concern the choice of MAX, β-values concern the choice of MIN.

Alpha-Beta-Search algorithm (see pseudocode in R&N):

- MAX nodes passes α values down:
  the max of the α- and v-values of the parent.
- MIN nodes passes β values down:
  the min of the β and v-values of the parent.

Alpha-beta Pruning: An Example



- Use Cutoff-Test(s) instead of Terminal-Test(s). E.g. limit by depth (search only to depth 5).
- Use Eval(s, p) instead of Utility(s, p). This is called an **evaluation function**.

Eval(s, p): How desirable is state s for player p?
E.g. an estimate of the chance of winning or the expected utility.

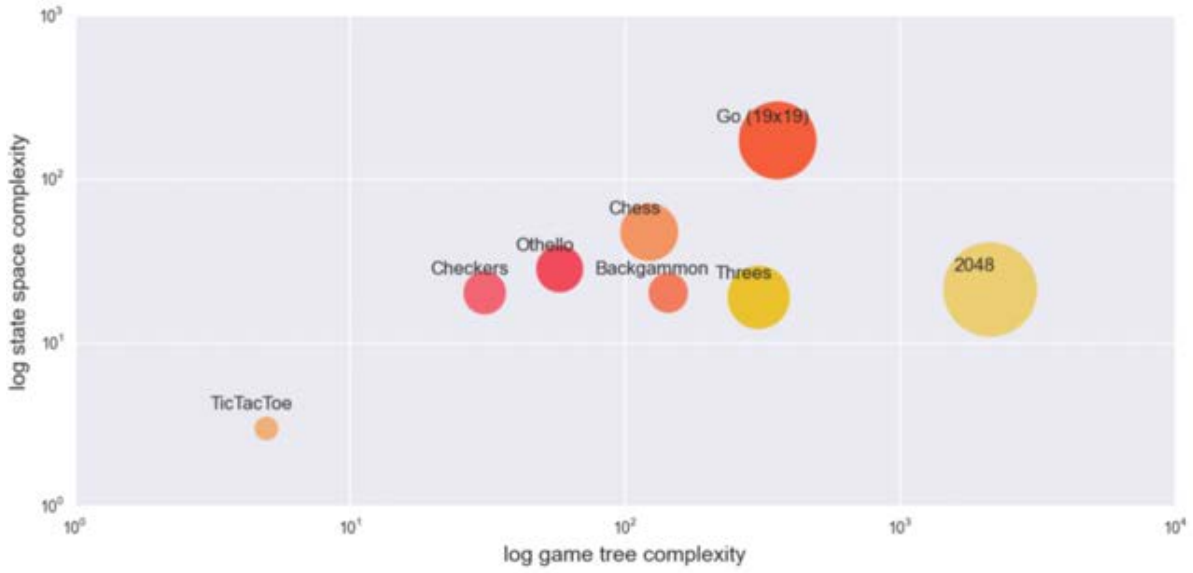
Expected utility. If 20% chance of getting utility 5 and 80% chance of getting utility 2, expected utility is $0.2 * 5 + 0.8 * 2 = 2.6$.

Evaluation functions in adversarial search play approximately the same role as heuristic functions in classical search: an estimate of "how good" the state is that can be used to guide the search/decisions.

$$\text{H-Minimax}(s, d) = \begin{cases} \text{Eval}(s) & \text{if Cutoff-Test}(s, d) \\ \max_{a \in \text{Actions}(s)} \text{Minimax}(\text{Result}(s, a), d+1) & \text{if Player}(s) = \text{max} \\ \min_{a \in \text{Actions}(s)} \text{Minimax}(\text{Result}(s, a), d+1) & \text{if Player}(s) = \text{min} \end{cases}$$

Games and Their Combinatorial Complexity

- State space complexity: number of possible states of the game.
- Game tree complexity: number of possible games.

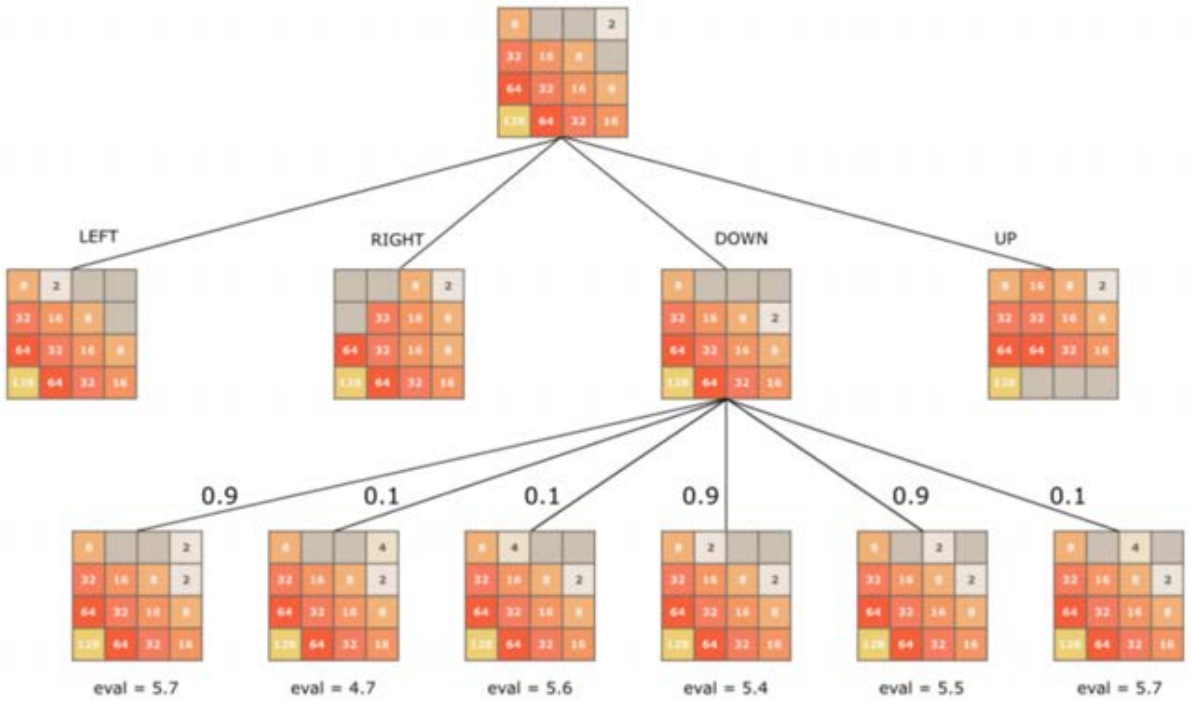


10^n on these axes mean that the complexity/size is $10^{(10^n)}$.

In 2048, the opponent is nature (the game engine), that makes probabilistic decisions, not maximising decisions. We call this player CHANCE. The best suited algorithm is EXPECTIMAX rather than MINIMAX.

$$\textsc{Expectimax}(s) = \begin{cases} \textsc{Utility}(s, \textsc{max}) & \text{if } \textsc{Terminal-Test}(s) \\ \max_{a \in \textsc{Actions}(s)} \textsc{Expectimax}(\textsc{Result}(s, a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \Sigma_{a \in \textsc{Actions}(s)} P(a, s) \cdot \textsc{Expectimax}(\textsc{Result}(s, a)) & \text{if } \textsc{Player}(s) = \textsc{chance} \end{cases}$$

Here $P(a, s)$ is the probability of CHANCE chosing action a in s.



Expectimax value of going down from root state:

$$\frac{0.9 \cdot (5.7 + 5.4 + 5.5) + 0.1 \cdot (4.7 + 5.6 + 5.7)}{3} = 5.512.$$

When designing a good evaluation function is very difficult
a (sample-based) learning approach can be used:
Monte-Carlo Tree Search (MCTS).

Monte-Carlo Tree Search is Minimax meets **reinforcement learning**:
The algorithm builds a partial game tree like Minimax,
states are evaluated by reinforcement learning.

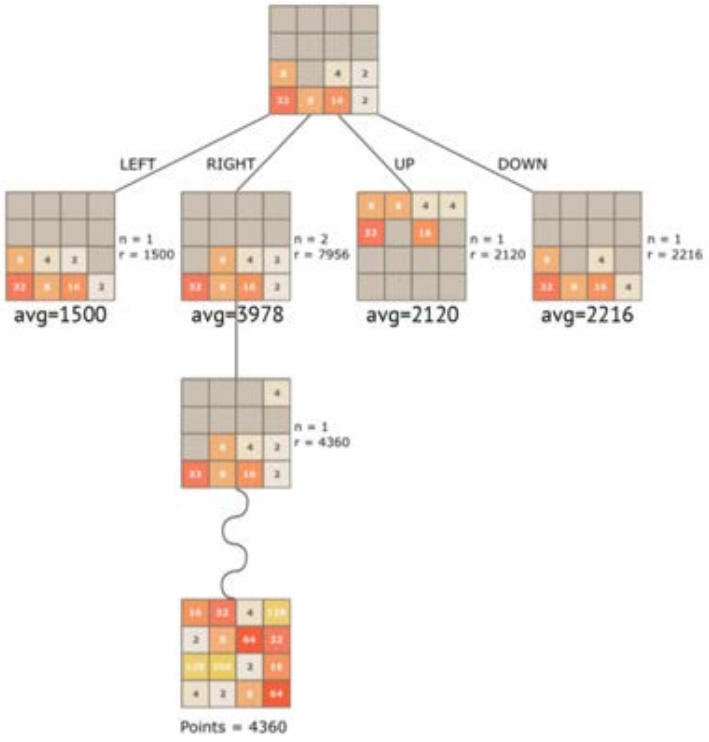
Differences to Minimax with cutoff:

- Non-uniform expansion of tree:
  most promising states expanded first.
- Evaluation is not hard-coded:
  based on Monte-Carlo sampling (random play-outs/simulations).

State evaluation

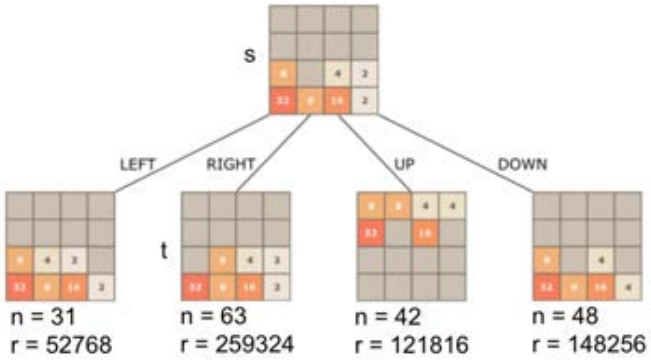
Evaluation of a state s (like in the Eval(s) in Minimax with cutoff):
Play n **random** full games beginning in s, and take the average utility of those games.
The utility is called a **reward** in Monte-Carlo Tree Search.

Example: State evaluation in 2048



- n: number of games played through the state.
- r: Sum of rewards (scores) received in those n games.

Example: State evaluation in 2048



- n: number of games played through the state.
- r: Sum of rewards (scores) received in those n games.

Node t chosen for **expansion** is node with max Upper Confidence Bound (UCB) value:

$$UCB(t) = \frac{r_t}{n_t} + \sqrt{\frac{2 \ln n_s}{n_t}}$$

where:

- s is parent of t,
- $\frac{r_t}{n_t}$ is the current average utility (reward) of t;
- n_t and n_s are the no of times t and s were played, resp.

Expanding a node in Tree- and Graph-search means adding all children.

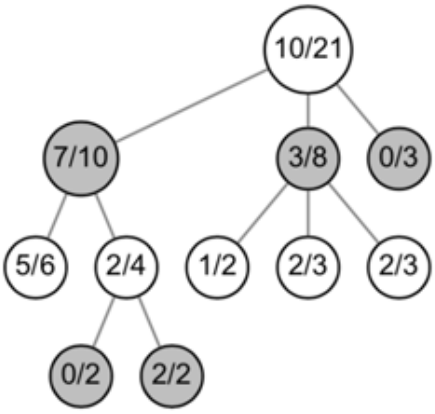
In MCTS, often *one child is added at a time*.
The **frontier** hence contains any node for which not all children have been generated.

Monte-Carlo Tree Search iteratively loops through the 4 steps:

1. **Selection**:
   a node in the frontier is selected for expansion:
   start at the root and find a path to a frontier node by iteratively selecting the best child.
   use UCB to find the best child of each node on the path.
2. **Expansion**:
   expand the selected node, that is add it to the frontier.
3. **Simulation**:
   play a random game from the generated child node;
   this is called a **playout** or **rollout**.
4. **Backup**:
   update the evaluation of all nodes on the path
   from the root to the generated child node based on the playout;
   this is called **backpropagation**.

MCTS example

- Two-player game: white and black.
- White moves in the white nodes.
- Each node is labelled by x/y:
  x - no of white's wins through that node
  y - no of games played through that node.
- Black is minimising, as in Minimax.



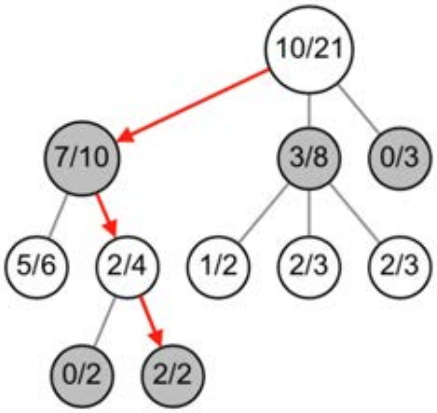
MCTS example

Step 1: Selection

- Two-player game: white and black.
- White moves in the white nodes.
- Each node is labelled by x/y:

  x - no of white's wins through that node

  y - no of games played through that node.
- Black is minimising, as in Minimax.



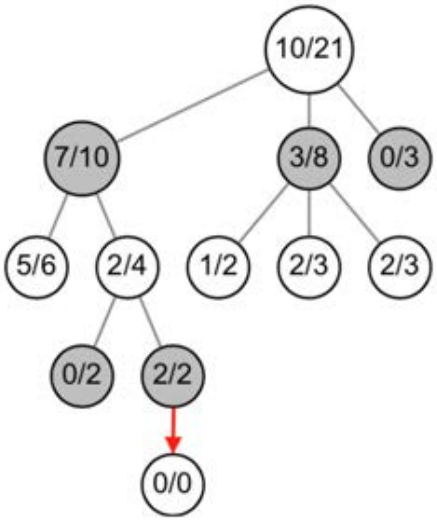
MCTS example

Step 2: Expansion

- Two-player game: white and black.
- White moves in the white nodes.
- Each node is labelled by x/y:

  x - no of white's wins through that node

  y - no of games played through that node.
- Black is minimising, as in Minimax.



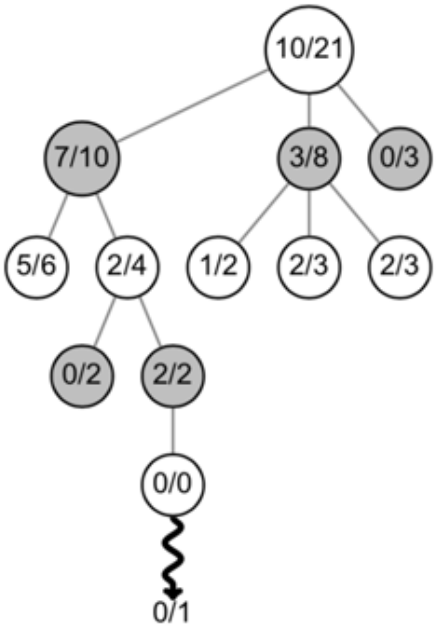
MCTS example

Step 3: Simulation

- Two-player game: white and black.
- White moves in the white nodes.
- Each node is labelled by x/y:

  x - no of white's wins through that node

  y - no of games played through that node.
- Black is minimising, as in Minimax.



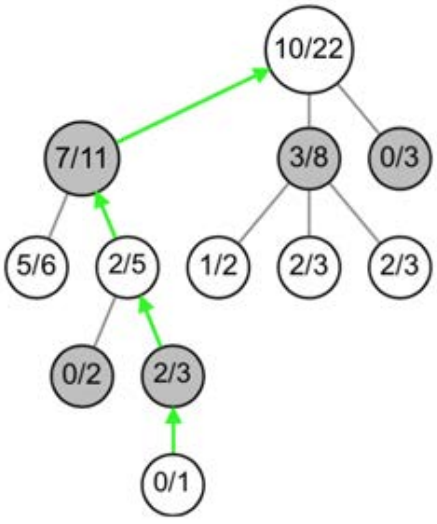
MCTS example

Step 4: Back-propagation

- Two-player game: white and black.
- White moves in the white nodes.
- Each node is labelled by x/y:

  x - no of white's wins through that node

  y - no of games played through that node.
- Black is minimising, as in Minimax.



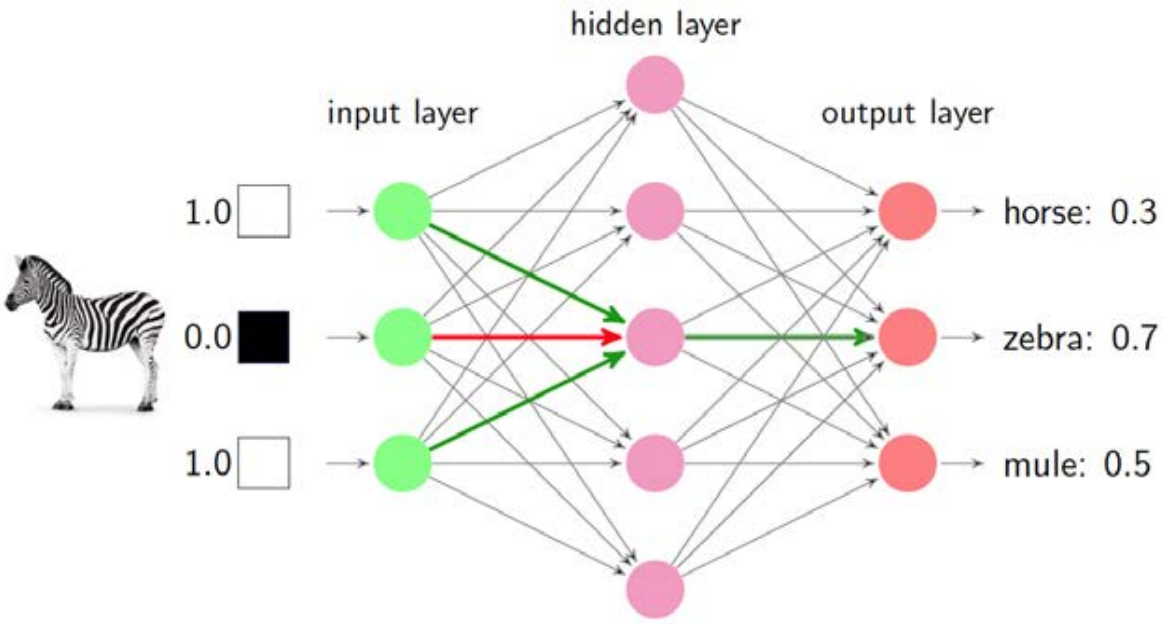
Monte-Carlo Tree Search converges to Minimax.

Monte-Carlo Tree Search is essentially just a more advanced way to prune the search tree than Minimax with cutoff.

For the game of Chess
MCTS is today far superior to Minimax with cut-offs and alpha-beta pruning.

For the game of 2048
Minimax with a good evaluation function performed better than MCTS.

Neural Networks



- Deep networks: many layers.
- Some layers are convolutional: sliding windows applying image filters to find geometrical features in different parts of the picture.

Three DTU students in a cafe

Three DTU students are sitting at a table.
The waiter asks: Does everyone want coffee?

- The first one says: I don't know.
- Then the second one says: I don't know.
- Then the third one says: No, not everyone wants coffee.
- The waiter gives the right people their coffees.

How?

Example

If you get Corona virus, you will get a cough.
You got a cough.

Therefore, you got Corona virus.

What do you think about this inference?

- You could be coughing even though you didn't get Corona virus!

Validity

Example

If you eat these cookies you will get sick.
You did not get sick.

Therefore, you didn't eat the cookies.

- This inference is **valid**!

Definition (Validity of Inference)

We call an inference *valid* if there is transmission of truth:
in every situation where all the premises are true,
the conclusion is also true.

In other words:
An inference is valid if it has no counter-examples (situations where the premises are all true while the conclusion is false).

Premise 1
Premise 2

Therefore, Conclusion

- If a premise is false, nothing follows about the conclusion!
- Validity only rules out the situation where *all* the premises are true and the conclusion is false!

What validity of inference says:

- If *all* premises are *true*, then the conclusion is *true*.
- If the conclusion is *false*, then *at least one* premise is *false*

Knowledge base (KB)

- Knowledge base is a set of **sentences**.
- Sentences are in a given **knowledge representation language**.
- Sentences represent assertions about the world.
- Some sentences have the status of **axioms** (they are given, must be and stay true).

- Adding to the knowledge base (TELL).
- Querying the knowledge base (ASK).
- Inference must obey the following rule:
  when one ASKs a question of the knowledge base, the answer should follow from what it has been told before (TELLed).

A generic knowledge-based agent

function KB-AGENT(*percept*) **returns** an *action*
persistent: *KB* a knowledge base, initially background knowledge;
t, a counter, initially 0 (time)

TELL(*KB*,MAKE-PERCEPT-SENTENCE(*percept, t*))
action $\leftarrow$ ASK (*KB*, MAKE-ACTION-QUERY(*t*))
TELL(*KB*,MAKE-ACTION-SENTENCE(*action, t*))
t $\leftarrow$ t+1
return *action*

- Syntax (rules for building sentences).
- Semantics (rules determining the truth of sentences).
- Model (or a possible world):

  if a sentence φ is true in a model m, we say that the m satisfies φ, and that m is a model of φ.

  $M(\varphi)$ stands for the set of all models of φ.
- Entailment between sentences:

  φ follows from ψ: $\psi \models \varphi$ iff $M(\psi) \subseteq M(\varphi)$.
- Logical inference is carried out with the use of entailment.

Basic Logical Notions, cntd.

- **Model-checking**: a procedure of logical inference by enumerating all models of KB and checking if the conclusion holds in all those models.
- If a logical inference algorithm can **derive** a sentence φ from KB, we write $KB \vdash \varphi$.
- A logical inference algorithm should be **sound** and **complete**.
- It should also be **grounded** in the underlying reality, e.g., KB should reflect the way the world really is, and adapt to the changing world.

not ...	$\neg$	negation
... and ...	$\wedge$	conjunction
... or ...	$\vee$	disjunction
if ... then ...	$\rightarrow$	implication
... if and only if ...	$\leftrightarrow$	bi-implication

A note on $\vee$: $\vee$ stands for the **inclusive** disjunction,
e.g.: "In order to pass the exam question 3 or question 4 must be answered correctly."
unlike "either ... or ..." in: "I will spend my summer in South America or in Asia."

Writing Natural Sentences in Logic

formalizing a sentence about a card game

Sentence "He has an Ace if he does not have a King or a Spade"
Formula $\neg(k \vee s) \rightarrow a$
k: "he has a King"
s : "he has a Spade"
a : "he has an Ace"

step-by-step

He has an Ace if he does not have a King or a Spade
if (he does not have a King or a Spade) then (he has an Ace)
(he does not have a King or a Spade) $\rightarrow$ (he has an Ace)
not (he has a King or a Spade) $\rightarrow$ (he has an Ace)
$\neg$(he has a King or a Spade) $\rightarrow$ (he has an Ace)
$\neg$ ((he has a King) or (he has a Spade)) $\rightarrow$ (he has an Ace)
$\neg$((he has a King) $\vee$ (he has a Spade)) $\rightarrow$ (he has an Ace)
$\neg(k \vee s) \rightarrow a$

Definition (Language of Propositional logic)

1. Every proposition letter ($p, q, r, ...$) is a formula.
2. If φ is a formula, then $\neg\varphi$ is also a formula.
3. If φ_1 and φ_2 are formulas, then $(\varphi_1 \wedge \varphi_2)$, $(\varphi_1 \vee \varphi_2)$, $(\varphi_1 \rightarrow \varphi_2)$ and $(\varphi_1 \leftrightarrow \varphi_2)$ are also formulas.
4. Nothing else is a formula.

Let Φ be a set of propositional letters, and let $p \in \Phi$ (that is: p is an element of the set Φ). Then the language of propositional logic is defined as follows:

$$\varphi := \quad p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \rightarrow \varphi) \mid (\varphi \leftrightarrow \varphi)$$

Well-formed formulas?

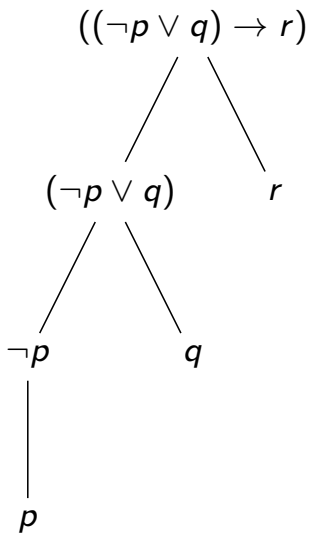
Let p be any proposition symbol from a set Φ,

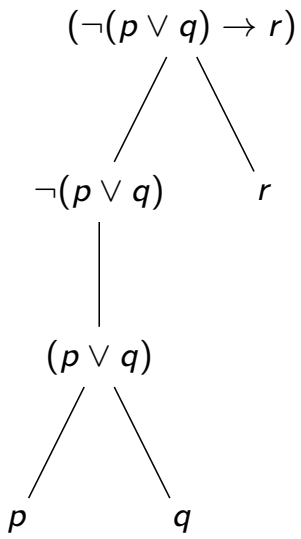
$$\varphi := \quad p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \rightarrow \varphi) \mid (\varphi \leftrightarrow \varphi)$$

Is this a formula of propositional logic?

$$(p \leftrightarrow (q \vee (r \rightarrow (p \vee \neg q))))$$

We can argue for it by providing its **syntactic derivation**:
$\varphi :=$
$\varphi \leftrightarrow \varphi :=$
$p \leftrightarrow \varphi :=$
$p \leftrightarrow (\varphi \vee \varphi) :=$
$p \leftrightarrow (q \vee \varphi) :=$
... $p \leftrightarrow (q \vee (r \rightarrow \varphi)) :=$
... $p \leftrightarrow (q \vee (r \rightarrow (p \vee \varphi))) :=$
$p \leftrightarrow (q \vee (r \rightarrow (p \vee \neg\varphi))) :=$
$p \leftrightarrow (q \vee (r \rightarrow (p \vee \neg q)))$.





Valuations

Propositions generate different **possibilities**, ways the actual world might be.

The set $\{p, q, r\}$ will generate $2^3 = 8$ possibilities:

p	q	r
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Each row is described by a function that assigns to each proposition a truth value: either 1 (="true") or 0 (="false"). We call such functions v, **valuations**.

For any proposition p, $v(p) = 1$ means that p is true, and $v(p) = 0$ means that p is false, in the situation represented by v.

φ	$\neg\varphi$
0	**1**
1	**0**

φ	ψ	$\varphi \wedge \psi$	$\varphi \vee \psi$	$\varphi \to \psi$	$\varphi \leftrightarrow \psi$
0	0	**0**	**0**	**1**	**1**
0	1	**0**	**1**	**1**	**0**
1	0	**0**	**1**	**0**	**0**
1	1	**1**	**1**	**1**	**1**

Tautologies of Propositional Logic

A **tautology** is a statement that is always true, i.e., it is true under any valuation.

Consider: $(p \to q) \vee (q \to p)$.

p	q	$(p \to q)$	$(q \to p)$	$(p \to q) \vee (q \to p)$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	1	1	1

A formula is **satisfiable** if there exists a valuation which makes the formula true.

Consider: $\neg(p \wedge q) \vee (\neg r)$.

p	q	r	$(p \wedge q)$	$\neg(p \wedge q)$	$\neg r$	$\neg(p \wedge q) \vee (\neg r)$
0	0	0	0	1	1	1
0	0	1	0	1	0	1
0	1	0	0	1	1	1
0	1	1	0	1	0	1
1	0	0	0	1	1	1
1	0	1	0	1	0	1
1	1	0	1	0	1	1
1	1	1	1	0	0	0

Logical Consequence

A formula ψ is a logical consequence of the set of formulas $\{\varphi_1, \ldots, \varphi_n\}$ if φ is always true when all of $\varphi_1, \ldots, \varphi_n$ are true.

In other words:
every valuation that makes all of $\varphi_1, \ldots, \varphi_n$ true, also makes ψ true.

When ψ is a logical consequence of $\varphi_1, \ldots, \varphi_n$ we write

$$\varphi_1, \ldots, \varphi_n \models \psi.$$

You can determine if $\varphi_1, \ldots, \varphi_n \models \psi$ using the following method:

1. Construct a truth table for all of the formulas $\varphi_1, \ldots, \varphi_n, \psi$.
2. Check that in every row where all of $\varphi_1, \ldots, \varphi_n$ get the value 1, ψ also gets the value 1.

Consider: $p, q \models p \vee q$.

p	q	$(p \wedge q)$	$(p \vee q)$
0	0	0	0
0	1	0	1
1	0	0	1
1	**1**	**1**	**1**

Proof system

A **proof system** is a set of formulas called **axioms** and a set of **rules of inference**.

A **proof** of a formula ψ is a sequence of formulas $\varphi_1, \ldots, \varphi_n$, with $\varphi_n = \psi$, such that each φ_k is either an axiom or it is derived from previous formulas by rules of inference.

When such a proof exists, we say that ψ is a **theorem** (of the system) and that ψ is **provable** (in the system), denoted by:

$$\vdash \psi$$

$\mathcal{H}$ is a proof system with three axiom schemes and one rule of inference.

For any formulas, φ, ψ and χ, the following formulas are axioms:

A1 $\vdash (\varphi \to (\psi \to \varphi))$
A2 $\vdash ((\varphi \to (\psi \to \chi) \to ((\varphi \to \psi) \to (\varphi \to \chi)))$
A3 $\vdash ((\neg\psi \to \neg\varphi) \to (\varphi \to \psi))$

The single inference rule of $\mathcal{H}$ is modus ponens (MP for short):

$$\frac{\vdash \varphi \qquad \vdash (\varphi \to \psi)}{\psi}\ \text{MP}$$

PROVING $\vdash (p \to p)$

Theorem. $\vdash (p \to p)$.

Proof.

1. $(p \to ((p \to p) \to p)) \to ((p \to (p \to p)) \to (p \to p))$ (A2)
2. $p \to ((p \to p) \to p)$ (A1)
3. $(p \to (p \to p)) \to (p \to p)$ (MP 1,2)
4. $p \to (p \to p)$ (A1)
5. $p \to p$ (MP 3,4)

The most intuitive way to check validity of inference by brut-force truth-tables.

- KB: Robert does well in the exam if and only if he is prepared or lucky. Robert does not do well in the exam.
- φ: Robert is not prepared.
- Question: $KB \models \varphi$?

Truth-table Method for Inference

The most intuitive way to check validity of inference by brut-force truth-tables.

- KB: Robert does well in the exam if and only if he is prepared or lucky. Robert does not do well in the exam.
- φ: Robert is not prepared.
- Question: $KB \models \varphi$?

r	p	ℓ	$p \vee \ell$	$r \leftrightarrow p \vee \ell$	$\neg r$	$\neg p$
0	0	0	0	**1**	**1**	**1**
0	0	1	1	0	**1**	1
0	1	0	1	0	**1**	0
0	1	1	1	0	**1**	0
1	0	0	0	0	0	1
1	0	1	1	**1**	0	1
1	1	0	1	**1**	0	0
1	1	1	1	**1**	0	0

- (Semantic) model checking:
  enumerating models and showing it holds in all models.
- (Syntactic) theorem proving:
  applying rules of inference directly to the sentences in our knowledge base to construct a proof of the desired sentence without consulting models.

Reason: If the number of models is large but the length of the proof is short, then theorem proving can be more efficient than model checking.

Some Crucial Concepts

- **Logical equivalence**
  two formulas φ and ψ are logically equivalent:
  if they are true in the same set of models, or
  if each of them entails the other: $\varphi \equiv \psi$ if and only if $\varphi \models \psi$ and $\psi \models \varphi$.
- **Validity (tautology)**
  φ is valid if it is true in all models.
  Every valid sentence is logically equivalent to $\top$.
- **Satisfiability**
  φ is satisfiable if it is true in some model.
  The SAT problem, determining the satisfiability of sentences, was the first problem shown to be NP-complete.

φ is valid iff $\neg\varphi$ is unsatisfiable

φ is satisfiable iff $\neg\varphi$ is not valid

Theorem (Semantic Deduction Theorem)
For any sentences φ and ψ of propositional logic

$\varphi \models \psi$ *if and only if* $\varphi \to \psi$ *is valid.*

$\varphi \models \psi$ if and only if the sentence $(\varphi \wedge \neg\psi)$ is
...
(reductio ad absurdum, proof by refutation, proof by contradiction)

Inference and Proofs

Inference rules are applied to derive a **proof of a formula**:

a chain of conclusions that leads to the formula.

Examples of inference rules

$$\frac{\varphi \to \psi,\ \varphi}{\psi}$$ (Modus Ponens)

$$\frac{\varphi \wedge \psi}{\varphi}$$ (And-Elimination)

A friendly statement of the theorem proving problem as a search problem:

- **Initial state**: the initial knowledge base.
- **Actions**: the inference rules.
- **Result**: the result of an action is to add the conclusion to KB.
- **Goal**: the sentence we are trying to prove.

Monotonicity

In **monotonic logics** the set of entailed sentences can only increase as information is added to the knowledge base.

For any sentences φ and ψ, if $KB \models \varphi$ then $KB \cup \psi \models \varphi$.

Note: **non-monotonic logics**, which violate the monotonicity property, capture a common property of human reasoning: changing one's mind.

$$\frac{\ell_1 \vee \ldots \vee \ell_k, m}{\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k} \qquad \text{(Unit Resolution)}$$

where literals ℓ_i and m are complementary (i.e., one is negation of the other).

$$\frac{\ell_1 \vee \ldots \vee \ell_k, m_1 \vee \ldots \vee m_n}{\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n} \qquad \text{(Full Resolution)}$$

where literals ℓ_i and m_j are complementary (i.e., one is negation of the other).

Practical comment: mind the **Factoring** (removing duplicates)!
E.g., if we resolve $(A \vee B)$ with $(A \vee \neg B)$, we obtain $(A \vee A)$.
Single A is enough!

Conjunctive normal form (CNF)

- Resolution applies to clauses (disjunctions of literals).
- So, KBs should then be coded as conjunctions of clauses (CNFs).
- Can any sentence of propositional logic be translated into CNF? **YES!**

Example

Robert will pass the exam if and only if he will be prepared or lucky.

$r \leftrightarrow p \vee s$

1. Eliminate $\leftrightarrow$: $(r \rightarrow p \vee s) \wedge (p \vee s \rightarrow r)$.
2. Eliminate $\rightarrow$: $(\neg r \vee p \vee s) \wedge (\neg(p \vee s) \vee r)$.
3. Move $\neg$ inwards (De Morgan): $(\neg r \vee p \vee s) \wedge ((\neg p \wedge \neg s) \vee r)$.
4. Distribute $\wedge$ over $\vee$: $(\neg r \vee p \vee s) \wedge (\neg p \vee r) \wedge (\neg s \vee r)$.

To show that $KB \models \varphi$, we show that $KB \wedge \neg\varphi$ is unsatisfiable.
We do this by proving a contradiction.

1. First, $KB \wedge \neg\varphi$ is converted into CNF.
2. Then, the resolution rule is applied to the resulting clauses.
3. Each pair that contains complementary literals is resolved to produce a new clause, which is added to the set if it is not already present.
4. The process continues until one of two things happens:
   - A there are no new clauses that can be added, in which case KB does not entail φ; or,
   - B two clauses resolve to yield the empty clause, in which case KB entails φ.

Example

- KB: Robert does well in the exam if and only if he is prepared or lucky. Robert does not do well the exam.
- φ: Robert is not prepared.
- Question: $KB \models \varphi$?

1. $(\neg r \vee p \vee s) \wedge (\neg p \vee r) \wedge (\neg s \vee r) \wedge \neg r$, formalisation of KB as CNF.
2. $(\neg r \vee p \vee s) \wedge (\neg p \vee r) \wedge (\neg s \vee r) \wedge \neg r \wedge p$, CNF for contradiction.
3. $clauses := (\neg r \vee p \vee s), (\neg p \vee r), (\neg s \vee r), \neg r, p$.
4. $clauses := clauses \cup \{r\}$ from resolving $(\neg p \vee r)$ with p.
5. $clauses := clauses \cup \{\bot\}$ from resolving r with $\neg r$.
6. Result: empty clause, so $KB \models \varphi$.

1. Always terminates.
2. Is complete, by the **ground resolution theorem**: If a set of clauses is unsatisfiable, then the resolution closure of those clauses contains the empty clause.

Special kinds of clauses

Sometimes it's enough if we restrict the language to special types of clauses:

Definition
A **definite clause** is a clause of literals of which exactly one is positive.
For example: $(\neg p \vee \neg s \vee r)$ is a definite clause, while $(p \vee s \vee \neg r)$ is not.

Definition
A **Horn clause** is a disjunction of literals of which at most one is positive.

All definite clauses are Horn clauses, as are clauses with no positive literals (**goal clauses**).

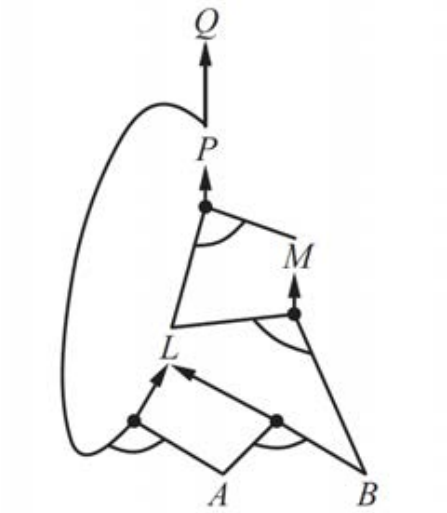
Horn clauses are closed under resolution:
if you resolve two Horn clauses, you get back a Horn clause.

Why are Horn clauses interesting?

- Definite clause can be written as an implication whose premise is a conjunction of positive literals and whose conclusion is a single positive literal. E.g., $(\neg p \vee \neg s \vee r)$ is equivalent to $(p \wedge s) \rightarrow r$. The premise is called the **body** and the conclusion is called the **head**. A sentence consisting of a single positive literal, such as r, is called a **fact** $(\top \rightarrow r)$.
- Inference with Horn clauses can be done by **forward-** and **backward-chaining**.
- Deciding entailment with Horn clauses is **linear** in the size of the knowledge base.

Forward- and Backward-chaining on AND-OR Graphs

$$
\begin{aligned}
&P \Rightarrow Q \\
&L \wedge M \Rightarrow P \\
&B \wedge L \Rightarrow M \\
&A \wedge P \Rightarrow L \\
&A \wedge B \Rightarrow L \\
&A \\
&B
\end{aligned}
$$



- the algorithms checking satisfiability: the SAT problem
- testing entailment $\varphi \models \psi$, is done by testing unsatisfiability of $\varphi \wedge \neg\psi$

Davis-Putnam Algorithm (DPLL Algorithm)

DPLL is an improvement on the truth-table method that works on CNFs.

Improvements over TT:

1. **Early termination**, e.g., if $(A \vee B) \wedge (A \vee C)$ is true if A is true, regardless of B and C.
2. **Pure symbol heuristic**, e.g., in $(A \vee \neg B)$, $(\neg B \vee \neg C)$, $(C \vee A)$, the symbol A is pure. If a sentence has a model, then it has a model with the pure symbols assigned so as to make their literals true, because doing so can never make a clause false.
3. **Unit clause heuristic**, e.g., if the model contains $B = \top$, then $(\neg B \vee \neg C)$ simplifies to $\neg C$, which is a unit clause. Assigning one unit clause can create another unit clause, such 'cascade' of forced assignments is called **unit propagation**.

function DPLL-SATISFIABLE?(s) **returns** *true* or *false*
  inputs: s, a sentence in propositional logic

  $clauses \leftarrow$ the set of clauses in the CNF representation of s
  $symbols \leftarrow$ a list of the proposition symbols in s
  return DPLL($clauses, symbols, \{\}$)

---

function DPLL($clauses, symbols, model$) **returns** *true* or *false*

  if every clause in $clauses$ is true in $model$ **then return** *true*
  if some clause in $clauses$ is false in $model$ **then return** *false*
  $P, value \leftarrow$ FIND-PURE-SYMBOL($symbols, clauses, model$)
  if P is non-null **then return** DPLL($clauses, symbols - P, model \cup \{P{=}value\}$)
  $P, value \leftarrow$ FIND-UNIT-CLAUSE($clauses, model$)
  if P is non-null **then return** DPLL($clauses, symbols - P, model \cup \{P{=}value\}$)
  $P \leftarrow$ FIRST($symbols$); $rest \leftarrow$ REST($symbols$)
  return DPLL($clauses, rest, model \cup \{P{=}true\}$) **or**
        DPLL($clauses, rest, model \cup \{P{=}false\}$))

WalkSat Algorithm

- Local search algorithms can be applied to SAT problem given the right evaluation function.
- The goal is to find an assignment that satisfies every clause, so
- evaluation function counts the number of unsatisfied clauses.
- We flip the truth value of one symbol at a time and escape local minima with randomness.

WalkSat: On every iteration, the algorithm picks an unsatisfied clause and picks a symbol in the clause to flip. It chooses randomly between two ways to pick which symbol to flip:

1. a **min-conflicts** step that minimises the number of unsatisfied clauses in the new state, and
2. a **random walk** step that picks the symbol randomly.

function WALKSAT(*clauses*, *p*, *max_flips*) **returns** a satisfying model or *failure*
  inputs: *clauses*, a set of clauses in propositional logic
    p, the probability of choosing to do a "random walk" move, typically around 0.5
    max_flips, number of flips allowed before giving up

  model ← a random assignment of *true*/*false* to the symbols in *clauses*
  for $i = 1$ **to** *max_flips* **do**
    if *model* satisfies *clauses* **then return** *model*
    clause ← a randomly selected clause from *clauses* that is false in *model*
    with probability *p* flip the value in *model* of a randomly selected symbol from *clause*
    else flip whichever symbol in *clause* maximizes the number of satisfied clauses
  return *failure*

The problem of Belief Revision

Belief revision is a topic of much interest in theoretical computer science and logic, and it forms a central problem in research into artificial intelligence. In simple terms: how do you update a database of knowledge in the light of new information? What if the new information is in conflict with something that was previously held to be true?

Gärdenfors, Belief Revision

We are talking about **beliefs** rather than **knowledge**.

In this context, the difference is that beliefs are changeable and can be false.

- belief := sentence
- belief := sentence in some formal language
- beliefs of an agent := a set of such sentences

Language of Beliefs in AGM

Beliefs are expressed in propositional logic:

- propositions $p, q, r, \ldots$
- connectives: negation ($\neg$), conjunction ($\wedge$), disjunction ($\vee$), implication ($\rightarrow$), and biconditional ($\leftrightarrow$).

Belief Sets

Example (What are the consequences of my beliefs?)

1. John is a bachelor. John is handsome. **John is a handsome bachelor.**
2. If we charge high fees for university, only the rich enroll. We charge high fees for university. **Only the rich enroll.**
3. The barber is male. The barber shaves only those men in town who do not shave themselves. **The barber is female.**

Belief set is a set of formulas that is **deductively closed**.
As such it is an abstract object.

Example
Assume Bob tells you that their beliefs include:

$$p, q.$$

Upon further inquiry it turns out that Bob also believes:

$$p \rightarrow \neg q.$$

Bob's beliefs are inconsistent!

Even though Bob's expressed beliefs do not include two complementary literals, $\neg q$ can be deduced from p and $p \rightarrow \neg q$.

He is committed to both q and $\neg q$, and so both are in Bob's belief set.

Bob revises his beliefs

Example
Assume Bob believes: $Cn(\{p, q, p \rightarrow q\})$
He learns, from a reliable source: $\neg q$
What should his new belief set be?

Bob's new belief set should be:
go to https://www.menti.com/ and enter code: 1362 5960

Option A: $Cn(\{p\})$
Option B: $Cn(\{p \rightarrow q, \neg q\})$
Option C: $Cn(\{p, q, p \rightarrow q\})$
Option D: $Cn(\{p, \neg q\})$
Option E: $Cn(\{p, p \rightarrow q\})$
Option F: $Cn(\{p, \neg q, p \rightarrow q\})$

Example

Assume Bob believes:

He learns, from a reliable source:

What should his new belief set be?

Bob's new belief set should be:

Option B: $Cn(\{p \rightarrow q, \neg q\})$

Option D: $Cn(\{p, \neg q\})$

Three Parts of Taking in New Information

What can I do to my belief set?

1. **Revision**: $B * \varphi$; φ is added and other things are removed, so that the resulting new belief set B' is consistent.
2. **Contraction**: $B \div \varphi$; φ is removed from B giving a new belief set B'.
3. **Expansion**: $B + \varphi$; φ is added to B giving a new belief set B'.

One formal way to combine those two is to use:

Levi identity

$$B * \varphi := (B \div \neg\varphi) + \varphi.$$

Belief revision can be defined as first removing any inconsistency with the incoming information and then adding the information itself.

Thinking in Terms of Plausibility Orders: Prior

Bob believes: $Cn(\{p, q, p \to q\})$, i.e., the state x is the most plausible.
But there are different ways in which the remaining options can be ordered.

p, q	$p, \bar{q}$	$\bar{p}, q$	$\bar{p}, \bar{q}$
			w
	y		
		z	
x			

p, q	$p, \bar{q}$	$\bar{p}, q$	$\bar{p}, \bar{q}$
	y		
		z	
			w
x			

In the above pictures, the lower the state the more plausible it is.

Bob believes: $Cn(\{p, q, p \rightarrow q\})$, i.e., the state x is the most plausible.

After revising with $\neg q$ his **posterior plausibility** changes differently depending on the **prior plausibility**.

We are looking for prior-minimal states that do not satisfy q.

p, q	$p, \bar{q}$	$\bar{p}, q$	$\bar{p}, \bar{q}$
			w
	y		
		z	
x			

Table: Option B: $Cn(\{p, \neg q\})$

p, q	$p, \bar{q}$	$\bar{p}, q$	$\bar{p}, \bar{q}$
	y		
		z	
			w
x			

Table: Option E: $Cn(\{p \rightarrow q, \neg q\})$

Bob believes: $Cn(\{p, q, p \rightarrow q\})$, i.e., the state x is the most plausible.

After contracting with q, Bob has to expand his view.

We are looking for prior-minimal states that do not satisfy q.

p, q	$p, \bar{q}$	$\bar{p}, q$	$\bar{p}, \bar{q}$
			w
	y		
		z	
x			

Table: $Cn(\{p\})$

p, q	$p, \bar{q}$	$\bar{p}, q$	$\bar{p}, \bar{q}$
	y		
		z	
			w
x			

Table: $Cn(\{p \leftrightarrow q\})$

After contraction Bob's beliefs are specified by the union of his prior most plausible world and the prior most plausible word not-entailing q.

p, q	$p, \bar{q}$	$\bar{p}, q$	$\bar{p}, \bar{q}$
		z	
			w
	y		
x			

more plausible ↓

Table: Plausibility order over valuations

Definition

Let P be a set of propositions (e.g. above, $P = \{p, q\}$). A **plausibility order** is a total preorder $\leq$ over the possible truth assignments W on P. A total preorder on X is a binary relation that is:

- transitive: for all $x, y, z \in X$, if $x \leq y$ and $y \leq z$, then $x \leq z$;
- complete: for all $x, y \in X$, $x \leq y$ or $y \leq x$.

Revision and Contraction on Plausibility Orders Formally

p, q	$p, \bar{q}$	$\bar{p}, q$	$\bar{p}, \bar{q}$
		z	
			w
	y		
x			

more plausible ↓

Table: B is determined by the most plausible world(s)

Let B be a belief set, φ a formula, and let $|\varphi| := \{x \in W \mid \varphi \text{ is true in } x\}$.

- $\varphi \in B$ iff $min_{\leq}(W) \subseteq |\varphi|$;

p, q	$p, \bar{q}$	$\bar{p}, q$	$\bar{p}, \bar{q}$
		z	
			w
	y		
x			

more plausible ↓

Table: $B * \neg p$ is determined by min world(s) with $\neg p$

Let B be a belief set, φ a formula, and let $|\varphi| := \{x \in W \mid \varphi \text{ is true in } x\}$.

- $\varphi \in B$ iff $min_{\leq}(W) \subseteq |\varphi|$;
- $\varphi \in B * \psi$ iff $min_{\leq}(|\psi|) \subseteq |\varphi|$;

Revision and Contraction on Plausibility Orders Formally

p, q	$p, \bar{q}$	$\bar{p}, q$	$\bar{p}, \bar{q}$
		z	
			w
	y		
x			

more plausible ↓

Table: $B \div \neg p$ is the union of the previous two

Let B be a belief set, φ a formula, and let $|\varphi| := \{x \in W \mid \varphi \text{ is true in } x\}$.

- $\varphi \in B$ iff $min_{\leq}(W) \subseteq |\varphi|$;
- $\varphi \in B * \psi$ iff $min_{\leq}(|\psi|) \subseteq |\varphi|$;
- $\varphi \in B \div \psi$ iff $min_{\leq}(|\neg\psi|) \cup min_{\leq}(W) \subseteq |\varphi|$

AGM$^{\div}$ Rationality Postulates of Contraction

1. **Closure**: $B \div \varphi = Cn(B \div \varphi)$
   the outcome is logically closed
2. **Success**: If $\varphi \notin Cn(\emptyset)$, then $\varphi \notin Cn(B \div \varphi)$
   the outcome does not contain φ
3. **Inclusion**: $B \div \varphi \subseteq B$
   the outcome is a subset of the original set
4. **Vacuity**: If $\varphi \notin Cn(B)$, then $B \div \varphi = B$
   if the incoming sentence is not in the original set then there is no effect
5. **Extensionality**: If $\varphi \leftrightarrow \psi \in Cn(\emptyset)$, then $B \div \varphi = B \div \psi$.
   the outcomes of contracting with equivalent sentences are the same
6. **Recovery**: $B \subseteq (B \div \varphi) + \varphi$.
   contraction leads to the loss of as few previous beliefs as possible
7. **Conjunctive inclusion**: If $\varphi \notin B \div (\varphi \wedge \psi)$, then $B \div (\varphi \wedge \psi) \subseteq B \div \varphi$.
8. **Conjunctive overlap**: $(B \div \varphi) \cap (B \div \psi) \subseteq B \div (\varphi \wedge \psi)$.

AGM* Rationality Postulates of Revision

1. **Closure**: $B * \varphi = Cn(B * \varphi)$
2. **Success**: $\varphi \in B * \varphi$
3. **Inclusion**: $B * \varphi \subseteq B + \varphi$
4. **Vacuity**: If $\neg\varphi \notin B$, then $B * \varphi = B + \varphi$
5. **Consistency**: $B * \varphi$ is consistent if φ is consistent.
6. **Extensionality**: If $(\varphi \leftrightarrow \psi) \in Cn(\emptyset)$, then $B * \varphi = B * \psi$.
7. **Superexpansion**: $B * (\varphi \wedge \psi) \subseteq (B * \varphi) + \psi$
8. **Subexpansion**: If $\neg\psi \notin B * \varphi$, then $(B * \varphi) + \psi \subseteq B * (\varphi \wedge \psi)$.

One formal way to combine those operations is to use:

Levi identity

$$B * \varphi := (B \div \neg\varphi) + \varphi.$$

Belief revision can be defined as first removing any inconsistency with the incoming information and then adding the information itself.

Contraction Remainders

Definition
For any set A and sentence φ the **remainder set** $A\perp\varphi$ is the set of inclusion-maximal subsets of A that do not imply φ.

Formally: a set C is an element of $A\perp\varphi$ just in case $C \subset A$, $C \not\models \varphi$, and there is no set C' such that $C' \not\models \varphi$ and $C \subset C' \subseteq A$.

So, in principle, the result of contraction **could be** chosen to be just one of the remainders, but...

So, in principle, the result of contraction **could be** chosen to be just one of the remainders...

But:
For belief set A and any φ inconsistent with A, A revised with φ will be complete, i.e., for any formula ψ, $\psi \in A * \varphi$ or $\neg\psi \in A * \varphi$.
This means that any agent after revising their belief becomes omniscient.

Argument:
Assume our agent has a belief set A.
Incoming information is φ, such that $\neg\varphi \in A$.

Then, for every formula ψ both $\neg\varphi \vee \psi$ and $\neg\varphi \vee \neg\psi$ are in A.
Upon contracting $\neg\varphi$ according to the above-described way,
for any formula ψ, either ψ or $\neg\psi$ are in $A \div \neg\varphi$.

Partial Meet Contraction

Solution: Let $B \div \varphi$ be the **intersection** of some of the remainders!

Definition
A **selection function** for B is a function γ such that if $B\perp\varphi$ is non-empty, then $\gamma(B\perp\varphi)$ is a non-empty subset of $B\perp\varphi$.

The outcome of the **partial meet contraction** is equal to the intersection of the selected elements of $B\perp\varphi$, i.e., $B \div \varphi = \cap\gamma(B\perp\varphi)$.

p, q	$p, \bar{q}$	$\bar{p}, q$	$\bar{p}, \bar{q}$
		z	
			w
	y		
x			

more plausible

Table: Plausibility order over valuations

Let B be a belief set, φ a formula, and let $|\varphi| := \{x \in W \mid \varphi \text{ is true in } x\}$.

Working with Sets of Formulas

Obviously, we want $B \div \varphi$ to be a subset of B that does not imply φ.

But carefully: **limit the removals** from B!

Definition
For any set A and sentence φ the **remainder for A and φ** is any set C s.t.:

- C is a subset of A,
- C does does not imply φ, and
- there is no set C' such that C' does not imply φ and $C \subset C' \subseteq A$.

Definition

A **belief base** is a set of sentences that is not necessarily deductively closed. Its elements represent beliefs that are held independently of any other belief or set of beliefs.

Those elements of the belief set that are not in the belief base are merely derived, i.e., they have no independent standing.

Changes are performed on the belief base. The underlying intuition is that the merely derived beliefs are not worth retaining for their own sake. If one of them loses the support that it had in basic beliefs, then it will be automatically and implicitly discarded.

Belief Bases and Belief Sets

For every belief base A, there is a belief set $Cn(A)$ that represents the beliefs according to A.
One belief set can be represented by different belief bases.
So, belief bases have more expressive power than belief sets.

Example

Alice's basic beliefs: p and q.
Bob's basic beliefs: p and $p \leftrightarrow q$.
Are Alice's and Bob's beliefs the same?
Go to: www.menti.com and enter code: 3983 4322

Alice's basic beliefs: p and q.
Bob's basic beliefs: p and $p \leftrightarrow q$.
Their **belief sets** with respect to p and q are the same.

So, Alice's and Bob's beliefs are **statically equivalent**.

On the other hand, they are not **dynamically equivalent**:

Say Alice and Bob receive and accept the information that p is false.
So, they both revise their belief bases to include the new belief $\neg p$.

After that, Alice has the basic beliefs $\neg p$ and q, whereas Bob has the basic beliefs $\neg p$ and $p \leftrightarrow q$.

Now, their belief sets are no longer the same:
Alice believes that q whereas Bob believes that $\neg q$.

Belief Base Contraction

Partial meet contraction, as defined before, is applicable to belief bases.

Note that $A \perp p$ is the set of maximal subsets of A that do not imply p;
it is still not sufficient that they do not contain p.

Example

$$\{p, p \wedge q, p \vee q, p \leftrightarrow q\} \perp p = \{\{p \vee q\}, \{p \leftrightarrow q\}\}$$

Propositional Logic:

Reasoning about situations using combinations of simple facts (with "and", "or", "not" etc). The *structure* of these facts was not analyzed further.

First-order Logic: First-order Logic looks at the internal structure of basic facts, especially, the objects that occur, the properties of these objects, and their relations to each other.

Example

j : John

a : Ann

Cxy : *x cooks for y.*	Cja : *John cooks for Ann.*
Vxy : *x visits y.*	Vaj : *Ann visits John.*
Hx : *x is happy.*	Hj : *John is happy.*

First-order Logic

- Reasoning about objects, predicates, and in principle, arbitrary forms of quantification.
- the most important system in logic today: it is a universal language for talking about **structure** (situation with objects, properties and relations).
  - Structures: friends on facebook, road systems, family trees, number systems
- First-order Logic has been used to increase precision in describing and studying all these structures, from linguistics and philosophy to computer science and mathematics.

The Language of First-order Logic: The basic vocabulary

- Names for **objects**:
  - variables $x, y, z, \ldots$ when the object is indefinite.
  - function symbols, e.g., constants ('proper names') $a, b, c, \ldots$,for special objects,
- **Properties and predicates** of objects:
  - Capital letters are predicate letters, with different numbers of arguments (arity):
  - 1-place predicates (unary predicates) are intransitive verbs ("walk") and common nouns ("boy"="being a boy"),
  - 2-place predicates are transitive verbs ("see")
  - 3-place predicates are so-called ditransitive verbs ("give")
  - ...
- **Sentence combination**:
  - The usual operators from propositional logic: $\neg, \vee, \wedge, \rightarrow, \iff$.
- **Quantification**:
  - Quantifiers: $\forall x$ ("for all x") and $\exists x$ ("there exists an x")

The grammar of First-order Logic

$$
\begin{array}{rcl}
\mathit{Sentence} & \rightarrow & \mathit{AtomicSentence} \mid \mathit{ComplexSentence} \\
\mathit{AtomicSentence} & \rightarrow & \mathit{Predicate} \mid \mathit{Predicate}(\mathit{Term}, \ldots) \mid \mathit{Term} = \mathit{Term} \\
\mathit{ComplexSentence} & \rightarrow & (\ \mathit{Sentence}\ ) \mid [\ \mathit{Sentence}\ ] \\
 & \mid & \neg\ \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \wedge \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \vee \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \Rightarrow \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \Leftrightarrow \mathit{Sentence} \\
 & \mid & \mathit{Quantifier}\ \mathit{Variable}, \ldots\ \mathit{Sentence} \\
\\
\mathit{Term} & \rightarrow & \mathit{Function}(\mathit{Term}, \ldots) \\
 & \mid & \mathit{Constant} \\
 & \mid & \mathit{Variable} \\
\\
\mathit{Quantifier} & \rightarrow & \forall \mid \exists \\
\mathit{Constant} & \rightarrow & A \mid X_1 \mid \mathit{John} \mid \cdots \\
\mathit{Variable} & \rightarrow & a \mid x \mid s \mid \cdots \\
\mathit{Predicate} & \rightarrow & \mathit{True} \mid \mathit{False} \mid \mathit{After} \mid \mathit{Loves} \mid \mathit{Raining} \mid \cdots \\
\mathit{Function} & \rightarrow & \mathit{Mother} \mid \mathit{LeftLeg} \mid \cdots
\end{array}
$$

Simple Statements about Objects

natural language	First-order Logic
Alex sleeps.	*Sa*
Alex is a dragon.	*Da*
He walks.	*Wx*
Alex eats Tweety.	*Eat*
John gives Mary the book.	*Gjmb*

Predicates in Mathematics

informally	logical/mathematical formula
42 is smaller than 303	$42 < 303$
x is smaller than 42	$x < 42$
y is even	$y \mid 2$
Point p lies between q and r	*Bpqr*

Propositional operators can be added in the obvious way to the preceding statements, and they function as before:

John does not see Mary.	$\neg Sjm$
Three is not less than two.	$\neg(3 < 2)$
Alex eats Tweety or Harry.	$Eat \vee Eah$
3 is less than 3 or 3 is less than 4.	$(3 < 3) \vee (3 < 4)$
x is odd	$\neg(2 \mid x)$
If John sees Mary, he is happy	$Sjm \rightarrow Hj$

Quantifiers

Existential: for saying that objects exist without naming them explicitly:

- Something happens: $\exists x Hx$

Universal: for saying that all objects satisfy a property:

- Everything is fun: $\forall x Fx$

- Some dragon walks: $\exists x(Dx \wedge Wx)$
- Alex loves a girl: $\exists x(Gx \wedge Sax)$
- A girl loves Alex: $\exists x(Gx \wedge Lxa)$
- A dragon bites itself: $\exists x(Dx \wedge Bxx)$

- Every boy walks: $\forall x(Bx \rightarrow Wx)$
- Every girl loves Alex: $\forall x(Gx \rightarrow Lxa)$

Nested Quantifiers

- Brothers are siblings: $\forall x \forall y\ (B(x, y) \rightarrow S(x, y))$
- Being a sibling is symmetric: $\forall x, y\ (S(x, y) \leftrightarrow S(y, x))$
- Everybody loves somebody: $\forall x \exists y\ L(x, y)$
- There is someone who is loved by everybody: $\exists y \forall x\ L(x, y)$
- Consider: $\forall x(M(x) \vee \exists x\ L(John, x))$: **confusing, use a fresh variable**

$$\forall x\ \neg\varphi \equiv \neg\exists x\ \varphi$$
$$\neg\forall x\ \varphi \equiv \exists x\ \neg\varphi$$
$$\forall x\ \varphi \equiv \neg\exists x\ \neg\varphi$$
$$\exists x\ \varphi \equiv \neg\forall x\ \neg\varphi$$

Formulas and Models

B: property of being a motorbike
C: property of being a cow
M: property of being a man
R: relation of riding

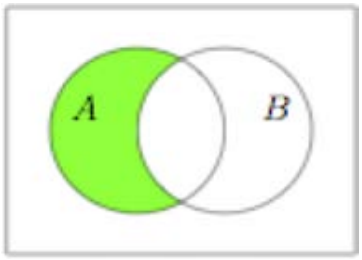
$$\exists x\exists y\exists z(((Mx \wedge Cy) \wedge Bz\ ) \wedge (Rxz \wedge Ryz))$$

A model in first-order logic consists of a set of objects and an interpretation that maps constant symbols to objects, predicate symbols to relations on those objects, and function symbols to functions on those objects.

Ontological Committments

Language	Ontological Commitment	Epistemological Commitment
Propositional	facts	true/false/unknown
First-order	facts, objects, relations	true/false/unknown
Temporal	facts, objects, relations, times	true/false/unknown
Probability	facts	degree of belief $\in [0, 1]$
Fuzzy	facts with degree of truth $\in [0, 1]$	known interval value

- Venn diagrams depict objects having certain properties (unary predicates).



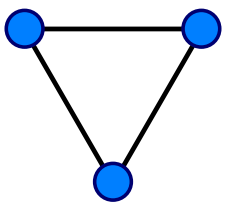
The green area is the set of individuals that make the formula $Ax \wedge \neg Bx$ true.

The variable x is not bound by quantifier. It is a **free variable**.

Pictures for more complex Formulas

- With Venn diagrams, we can reason about sets of objects with certain *properties*.
- But, if we want to talk about *relations* between objects, we need another kind of pictures: Graphs!

Example



R : relation of being linked by an edge
True in this situation: $\forall x \exists y Rxy$, $\forall x \neg Rxx$
False: $\exists x \forall y Rxy$ $\exists x \exists y (Rxy \wedge \neg Ryx)$

- First-order Logic is more powerful if we can talk about things being equal.
- Equality can be expressed with the relation $=$.

Example
Olivia loves Peter but Peter loves another girl.

$$Lop \wedge \exists x((Gx \wedge \neg(x = o)) \wedge Lpx)$$
$$Lop \wedge \exists x((Gx \wedge x \neq o) \wedge Lpx)$$

Expressing Uniqueness

We can use equality in order to express uniqueness.

Example
Let P be the property of being a pope.

There is exactly one individual that has the property of being a pope.

$$\exists x(Px \wedge \forall y(Py \rightarrow (y = x)))$$

Similar to uniqueness, we can also express that there is a certain number of individuals that have some property.

Example

John has two sisters.

$$\exists x \exists y (((Sxj \wedge Syj) \wedge x \neq y) \wedge \forall z (Szj \rightarrow ((z = x) \vee (z = y))))$$

where S : relation of being a sister of someone.

This formula says that there are two individuals which are different and both are sisters of John, and every individual that is a sister of John has to be one of those two.

An Alternative Semantics

John has two sisters, Ann and Barbara.

$$Sister(John, Ann) \wedge Sister(John, Barbara)$$

$$Sister(John, Ann) \wedge Sister(John, Barbara) \wedge \ Ann \neq Barbara$$
$$\wedge \ \forall x \ Sister(John, x) \rightarrow (x = Ann \ \vee \ x = Barbara)$$

Too cumbersome, so in **database semantics**, the following are assumed:

1. unique names assumption;
2. closed-world assumption;
3. domain closure.

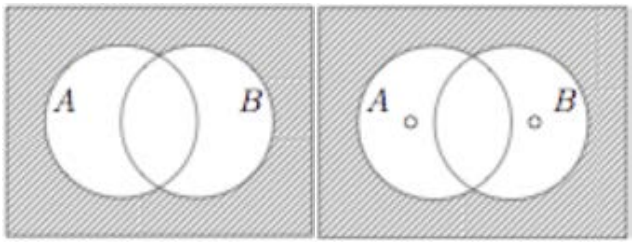
1. Identify the task.
2. Assemble the relevant knowledge.
3. Decide on a vocabulary (ontology).
4. Encode general knowledge about the domain.
5. Encode a description of the specific problem instance.
6. Pose queries to the inference procedure and get answers.
7. Debug the knowledge base.

See the textbook (Chapter 9 of R&N) for the **electronic circuit** domain.

Semantic Example: Venn Diagrams

If we are only concerned with unary predicates, we can use Venn diagrams to determine if an *inference* of First-order Logic is *valid*.

Example
Is it the case that $\forall x(Ax \vee Bx) \models \forall x\ Ax \vee \forall x\ Bx$?



No! $\forall x(Ax \vee Bx) \not\models \forall x\ Ax \vee \forall x\ Bx$

Finding a model for which the premiss is true and the conclusion false!

Let us now look at ways to derive new formulas from old formulas.

From:

$$\forall x\ (King(x) \wedge Greedy(x) \rightarrow Evil(x))$$

it follows that:

$$King(John) \wedge Greedy(John) \rightarrow Evil(John)$$
$$King(Richard) \wedge Greedy(Richard) \rightarrow Evil(Richard)$$
$$King(Father(John)) \wedge Greedy(Father(John)) \rightarrow Evil(Father(John))$$

Inference Rules for Quantifiers: Universal Instantiation

Universal Instantiation (UI)
infer any sentence obtained by substituting a ground term (a term without variables) for the variable.

Formally, let $SUBST(\theta, \alpha)$ be the result of applying θ to α.

$$\frac{\forall v\ \alpha}{SUBST(\{v/g\}, \alpha)}$$

for any variable v and ground term g.

For example, we got the three sentences above via $\{x/John\}$, $\{x/Richard\}$, and $\{x/Father(John)\}$.

Existential Instantiation (EI)
replace the variable by a single new constant symbol.

Formally, for any sentence α, variable v, and constant symbol k that does not appear elsewhere in the knowledge base:

$$\frac{\exists v\ \alpha}{SUBST(\{v/k\}, \alpha)}$$

For example, from the sentence $\exists x\ Crown(x) \wedge OnHead(x, John)$ we can infer the sentence $Crown(C1) \wedge OnHead(C1, John)$, as long as C1 does not appear elsewhere in the knowledge base.

In logic, the new name is called a **Skolem constant**.

Inference Rules for Quantifiers: Discussion

- UI can be applied many times to produce different consequences
- EI can be applied once, then the existential sentence is discarded
- the new knowledge base is not logically equivalent to the old
- but it can be shown to be inferentially equivalent:
- it is satisfiable exactly when the original knowledge base is satisfiable

Propositionalisation gives completeness: any entailed sentence can be proved.

But we do not know until the proof is done that the sentence is entailed!

What happens when the sentence is not entailed? Can we tell?

In FOL we cannot: proof can go on and on and we'll never know if it'll stop. Just like the **halting problem for Turing machines**. See here: movie

A first-order inference rule: Generalized Modus Ponens

Propositionalization approach is rather inefficient.

For example, given the query $Evil(John)$ and the knowledge base:

$$\forall x\ King(x) \wedge Greedy(x) \rightarrow Evil(x)$$
$$King(John)$$
$$Greedy(John)$$
$$Brother(Richard, John)$$

why would we generate all the instantiations of the first sentence?

Instead, we'd prefer to conclude $Evil(John)$ directly.

For atomic sentences p_i, p_i', and q,
if there is a substitution θ s.t. $SUBST(\theta, p_i') = SUBST(\theta, p_i)$, for all i:

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n) \rightarrow q)}{SUBST(\theta, q)}$$

Generalized Modus Ponens is a **lifted** version of Modus Ponens:
raised from ground (variable-free) propositional logic to first-order logic.

Analogously, lifted versions of the forward chaining, backward chaining, and resolution algorithms can be provided (Chapter 9 of R&N).

Unification

Lifted inference requires substitutions that make different expressions identical.

Unification and is a key component of all first-order inference algorithms.
The UNIFY algorithm takes two sentences and returns a unifier (if exists):

$$UNIFY(p, q) = \theta \text{ where } SUBST(\theta, p) = SUBST(\theta, q)$$

Query: $AskVars(Knows(John, x))$: whom does John know?

To answer we need to find all sentences in KB that unify with $Knows(John, x)$.

For example:

$$UNIFY(Knows(John, x), Knows(John, Jane)) = \{x/Jane\}$$
$$UNIFY(Knows(John, x), Knows(y, Bill)) = \{x/Bill, y/John\}$$
$$UNIFY(Knows(John, x), Knows(y, Mother(y))) = \{y/John, x/Mother(John)\}$$
$$UNIFY(Knows(John, x), Knows(x, Elizabeth)) = fail$$

Last unification can be solved by introducing a new variable:
standarizing apart one of the sentences.

Most General Unifier

$UNIFY(Knows(John, x), Knows(y, z))$ could return:
$\{y/John, x/z\}$ or $\{y/John, x/John, z/John\}$

Which is better? First gives $Knows(John, z)$, second $Knows(John, John)$.
The first one is more general!

Theorem:
For every unifiable pair of expressions, there is a unique **most general unifier** (MGU).

As before, resolution requires conjunctive normal form (CNF).

In FOL literals can contain variables, assumed to be universally quantified:

$$\forall x, y, z\ American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \rightarrow Criminal(x)$$

becomes, in CNF:

$$\neg American(x) \vee \neg Weapon(y) \vee \neg Sells(x, y, z) \vee \neg Hostile(z) \vee Criminal(x)$$

Every FOL sentence can be converted into an inferentially equivalent CNF.

The CNF sentence will be unsatisfiable just when the original is too,
so we can do proofs by contradiction on the CNF sentences.

Converting to CNF in FOL

Everyone who loves all animals is loved by someone.

$$\forall x\ (\forall y\ (Animal(y) \rightarrow Loves(x, y)) \rightarrow \exists y\ Loves(y, x))$$

- $\forall x\ (\forall y\ (Animal(y) \rightarrow Loves(x, y)) \rightarrow \exists y\ Loves(y, x))$
- **Eliminate implication**:
  $\forall x\ (\neg\forall y\ (\neg Animal(y) \vee Loves(x, y)) \vee \exists y\ Loves(y, x))$
- **Move $\neg$ inwards**:
  - $\forall x\ (\exists y\ (\neg(\neg Animal(y) \vee Loves(x, y))) \vee \exists y\ Loves(y, x))$
  - $\forall x\ (\exists y\ (\neg\neg Animal(y) \wedge \neg Loves(x, y)) \vee \exists y\ Loves(y, x))$
  - $\forall x\ (\exists y\ (Animal(y) \wedge \neg Loves(x, y)) \vee \exists y\ Loves(y, x))$
- **Standarize variables**: $\forall x\ (\exists y\ Animal(y) \wedge \neg Loves(x, y)) \vee \exists z\ Loves(z, x))$
- **Skolemize (EI)**: $\forall x\ ((Animal(A) \wedge \neg Loves(x, A)) \vee Loves(B, x))$ **NO!**
  $\forall x\ ((Animal(F(x)) \wedge \neg Loves(x, F(x))) \vee Loves(G(x), x))$
- **Drop universal quantifiers**:
  $(Animal(F(x)) \wedge \neg Loves(x, F(x))) \vee Loves(G(x), x)$
- **Distribute $\vee$ over $\wedge$**:
  $(Animal(F(x)) \vee Loves(G(x), x)) \wedge (\neg Loves(x, F(x)) \vee Loves(G(x), x))$!

Resolution Rule for FOL

Two standardized apart clauses are resolved if they have complementary literals.

FOL literals are complementary if one unifies with the negation of the other.

$$\frac{\ell_1 \vee \ldots \vee \ell_k, m_1 \vee \ldots \vee m_n,}{SUBST(\theta, \ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)}$$

where $UNIFY(\ell_i, \neg m_j) = \theta$.

Full resolution, which extends the above to unifying sets of literals is a complete inference procedure for FOL.